BEA AquaLogic Service Bus™

AquaLogic Service Bus Interoperability Solutions Guide

Version: 2.1
Document Revised: December 2005

Contents

Introduction

AquaLogic Service Bus provides a unified software product for implementing and deploying your Service-Oriented Architecture (SOA).

AquaLogic Service Bus supports broad compliance with messaging standards including SOAP 1.1, HTTP, JMS, SMTP/POP/IMAP, FTP, SSL, XML 1.0, XML Schema, WSDL 1.1, WSRP 1.0, and WS-Security.

This section contains information about AquaLogic Service Bus interoperability. It includes the following topics:

- Summary of Interoperability
- FTP and Email Servers
- Security Providers
- Web Service Standards
- HTTP Standards
- XPath and XQuery
- JMS
- Databases
- Platform Interoperability

Summary of Interoperability

The following table summarizes the versions of the platforms, standards, FTP servers, and so on, that are certified to interoperate with AquaLogic Service Bus.

Table 1-1 AquaLogic Service Bus Interoperability Matrix

AquaLogic Service Bus Interoperates With . . .	Version . . .	For More Information . . .
FTP and Email Servers	• Microsoft Windows IIS • Sol/Apache	"FTP and Email Servers" on page 1-4
Security Providers	• WebLogic Server 9.x (for providers, except XACML authorization provider) • WebLogic Server 9.1 XACML authorization provider	"Security Providers" on page 1-4
Web Service Standards	• SOAP 1.1 • WSDL 1.1 • WebLogic Platform 8.1 SP4 or later (except security) • WS-Security using WebLogic Server 9.x • WebLogic Server 9.x WS-Policy • WS-I	"Web Service Standards" on page 1-5
HTTP	• 1.0	"HTTP Standards" on page 1-5
XQuery	• 1.0	"XPath and XQuery" on page 1-6
XPath	• 2.0	
JMS	• WebLogic Server 9.x JMS • IBM WebSphere MQ/JMS 5.3	"JMS" on page 1-6

AquaLogic Service Bus Interoperates With . . .	Version . . .	For More Information . . .
Databases	• Oracle 9.2.0.4 and later patch sets of 9.2.x • Oracle 10.1.0.4 and later patch sets of 10.1.x • PointBase 5.1 (in development mode, and non cluster environments only) • DB2 8.2 FixPak2 (equivalent to 8.1 FixPak 9) and later FixPaks • SQL Server 2000 SP3+ • Sybase 12.5.03 and later patch levels of 12.5.x	. . about databases and drivers, see Supported Database Configurations in *Supported Configurations for AquaLogic Service Bus*
Microsoft .NET	• 1.1	"Platform Interoperability" on page 1-6
Apache Axis	• 1.2.1	
WebLogic JMS	• WebLogic Platform 8.1 SP4 or later	
WebLogic Platform	• WebLogic Platform 8.1 SP4 or later	
WebLogic Integration	• WebLogic Integration 8.1 SP4 or later	
IBM WebSphere MQ	• 6.0	
AquaLogic Service Registry	• 2.0	
Web Services for Remote Portlets (WSRP)	• 1.0	
BEA Tuxedo/WebLogic Tuxedo Connector	• 8.1/9.0	

FTP and Email Servers

AquaLogic Service Bus is certified against the following FTP and Email servers:

- FTP
  - Microsoft Windows IIS FTP Server
  - Sol/Apache FTP Server
- Email
  - Microsoft Windows IIS SMTP Server
  - Sol/Apache SMTP Server

Security Providers

AquaLogic Service Bus is certified against the following security providers:

- WebLogic Server 9.x default authentication provider
- WebLogic Server 9.x default authorization provider
- WebLogic Server 9.x default credential mapper
- WebLogic Server 9.x PKI credential mapper
- WebLogic Server 9.x PKI credential provider
- WebLogic Server 9.x Java KeyStores (JKS)
- WebLogic Server 9.x default User Name Token and X509 Token handlers
- WebLogic Server 9.1 XACML authorization provider
- WebLogic Server 9.1 XACML Role Mapping provider
- WebLogic SAML Identity Assertion Provider V2
- WebLogic SAML Credential Mapping Provider V2

For more information about managing and configuring AquaLogic Service Bus security, see in Securing Inbound and Outbound Messages in the *BEA AquaLogic Service Bus User Guide.*

Note: The defaults for WebLogic Server 9.1 are the XACML Authorization provider and XACML Role Mapping provider.

Web Service Standards

AquaLogic Service Bus is certified against the following Web Service standards:

- SOAP 1.1
- WSDL 1.1
- WS-Addressing
- WS-Security using WebLogic Server 9.x WS client or WebLogic Server 9.x business services.
- WSDL attached WS-Policies as supported by WebLogic Server 9.x.
- WebLogic Server 9.x WS-Policy—identity, integrity, confidentiality and timestamp assertions.

  Note: WS-ReliableMessaging assertions are not supported.

WS-I

AquaLogic Service Bus includes WS-I compliance. However, in some cases, AquaLogic Service Bus does not reject SOAP/HTTP messages that are not WS-I compliant. This enables you to build implementations with service endpoints which are not strictly WS-I compliant.

When you configure a proxy service or business service, you can use the AquaLogic Service Bus Console to specify whether you want AquaLogic Service Bus to enforce WS-I compliance for the service. When you configure WS-I compliance for a proxy service, WS-I compliance checks are performed when the proxy service receives a message as a response from an invoked service with a Service Callout, a route node, or on a proxy service.

For information about the types of messages to which the compliance checks are applied and the nature of those checks, see "WS-I Compliance" in Modeling Message Flow in AquaLogic Service Bus in the *BEA AquaLogic Service Bus User Guide*.

HTTP Standards

AquaLogic Service Bus is certified against the following HTTP protocols:

- HTTP 1.0
- Transport-Layer Security (TLS) Secure Sockets Layer (/SSL) protocols

  Note: TLS/SSL support is the same as for WebLogic Server 9.x

XPath and XQuery

AquaLogic Service Bus is certified against the following protocols:

- BEA's implementation of XQuery 1.0

  To learn more, see XQuery Implementation in *BEA AquaLogic Service Bus User Guide*.

- BEA's implementation of XPath 2.0

JMS

AquaLogic Service Bus is certified against the following JMS implementations:

- WebLogic Server 9.x JMS

  For information about JMS interoperability, see Chapter 3, "Interoperability with JMS."

- IBM WebSphere MQ/JMS 5.3

  For information about AquaLogic Service Bus and MQ/JMS interoperability, see Chapter 4, "Interoperability with WebSphere MQ."

Databases

For complete information about supported databases and drivers, see Supported Database Configurations in *Supported Configurations for AquaLogic Service Bus*.

Platform Interoperability

AquaLogic Service Bus is certified to interoperate with the following platforms:

- WS-* and JMS interoperability with WebLogic Platform 8.1 SP4 or later (except for WS-Security)

  For information about AquaLogic Service Bus and JMS interoperability, see Chapter 3, "Interoperability with JMS" and Chapter 4, "Interoperability with WebSphere MQ."

- Web Services for Remote Portlets (WSRP) with BEA WebLogic Portal

  For information about AquaLogic Service Bus and WSRP interoperability, see Chapter 6, "Interoperability with Web Services for Remote Portlets (WSRP)."

- MQ event generator and control in WebLogic Integration 8.1 SP4 or later

- BEA Tuxedo

  For information about AquaLogic Service Bus and BEA Tuxedo interoperability, see Chapter 2, "Interoperability with BEA Tuxedo."

- BEA AquaLogic Service Registry 2.0

  To learn about using AquaLogic Service Registry with AquaLogic Service Bus, see UDDI in the *BEA AquaLogic Service Bus User Guide*. To learn about AquaLogic Service Registry, see the product documentation at the following URL:

  `http://e-docs.bea.com/alsr/docs20/`

- Microsoft .NET 1.1

  Style-encoding: document-literal, rpc-encoded.

  – AquaLogic Service Bus supports document-literal and interoperates with .NET services.

  – AquaLogic Service Bus interoperates with .NET rpc-encoded services in cases of inbound and outbound (routing/publish). In these cases, interoperability is possible regardless of parameter types.

  – AquaLogic Service Bus Service Callouts may fail to interoperate with .NET rpc-encoded services.

  Note: DIME attachments is not supported by AquaLogic Service Bus.

- Apache Axis 1.2.1

- IBM WebSphere MQ 6.0

AquaLogic Service Bus is not certified to interoperate with the following platforms:

- WebLogic Platform 7.0

See the *BEA AquaLogic Service Bus Release Notes* for the latest information about patches or updates that may be required to support your interoperability scenarios.

Interoperability with BEA Tuxedo

Introduction

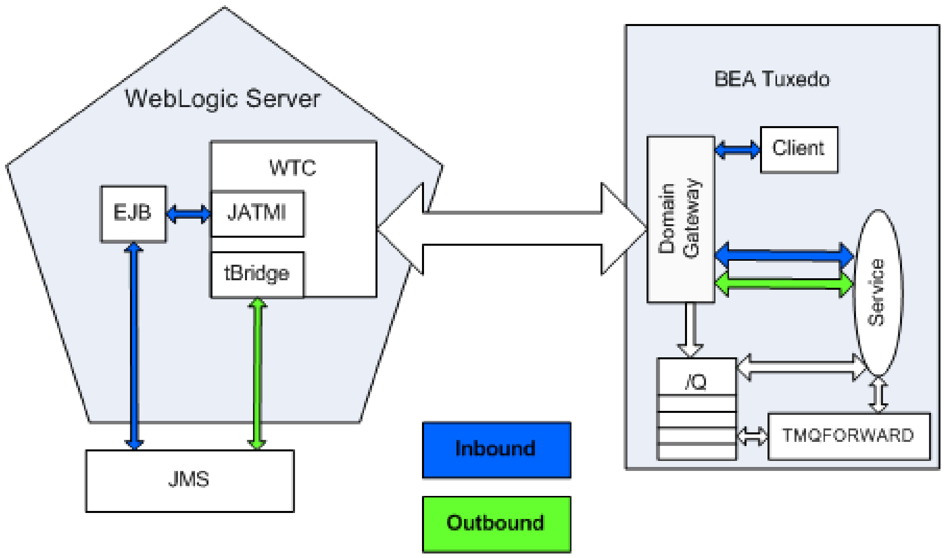
BEA AquaLogic Service Bus and BEA Tuxedo can interoperate to use the services that each product offers.

- When AquaLogic Service Bus uses services offered by BEA Tuxedo, a request for a Tuxedo service can be placed on a JMS queue and the reply to that request can be received from another JMS queue. The term “outbound” refers to this scenario.
- When BEA Tuxedo uses services offered by AquaLogic Service Bus, BEA Tuxedo services can call an EJB as though it were another BEA Tuxedo application. The term “inbound” refers to this scenario.

You can use the following related documentation to learn more about this environment:

- For information on setting up your AquaLogic Service Bus environment, see the *AquaLogic Service Bus Administration Console Online Help*.
- For information on setting up your BEA Tuxedo environment, see *Setting Up a Tuxedo Application* in the BEA Tuxedo documentation.
- For information about the WebLogic Tuxedo Connector, see WebLogic Tuxedo Connector in the WebLogic Server documentation.
- If you want to set up and use a working example of this interoperability scenario, see “Interoperability with AquaLogic Service Bus and Tuxedo” on the dev2dev web site.

The following diagram summarizes this message handling process:



This chapter includes the following sections:

- Using BEA Tuxedo Services from AquaLogic Service Bus (Outbound Example)
- Using AquaLogic Service Bus Services from BEA Tuxedo (Inbound Example)

Using BEA Tuxedo Services from AquaLogic Service Bus (Outbound Example)

The following sections describe how to use BEA Tuxedo services from AquaLogic Service Bus:

- Implementation Overview
- Before You Begin
- Configuring WebLogic Tuxedo Connector and the Tuxedo Queuing Bridge
- Configuring a New Business Service
- Testing Your Configuration

Implementation Overview

AquaLogic Service Bus can utilize services offered by BEA Tuxedo using WebLogic Tuxedo Connector. WebLogic Tuxedo Connector provides a JMS bridge (tBridge) that can directly call Tuxedo services.

After you configure WebLogic Tuxedo Connector and the Tuxedo Queuing Bridge, a request for a Tuxedo service can be placed on a JMS queue and the reply to that request can be received from another JMS queue. tBridge handles the conversion from the JMS message to a Tuxedo buffer type, calls the imported Tuxedo service, converts the reply buffer back to a JMS message type, and places the converted reply onto a JMS queue. The request to Tuxedo can either be made directly to a service or placed on a Tuxedo /Q queue.

Before You Begin

Gather the following information about the Tuxedo application that AquaLogic Service Bus will use:

- ID of the Tuxedo local access point.
- Network address of the Tuxedo local access point.
- Name of the exported Tuxedo service.
- Whether the service needs XML-to-FML and FML-to-XML conversion.

  If XML-to-FML and FML-to-XML conversion is needed, you will need to add one or more Field Table classes to a resource section in the WebLogic Tuxedo Connector configuration. This task is described in "Create Field Table Classes (if Required)" on page 2-8.

  The example described in the following sections assumes the use of FML/FML32 buffer types.
- ID of the access point that the Tuxedo domain gateway will use to refer to this WebLogic Tuxedo Connector instance.
- Network address that the Tuxedo domain gateway has defined for this WebLogic Tuxedo Connector Local Access Point.

Prior to configuring the Tuxedo Queuing Bridge, you must create several JMS queues if they do not already exist. Queues are required for the following purposes:

- One or more queues on which to place requests to the Tuxedo service

- A queue to pick up replies from the Tuxedo service
- A queue to receive messages that failed to be delivered to Tuxedo
- A queue to receive error replies from the called service

Configuring WebLogic Tuxedo Connector and the Tuxedo Queuing Bridge

You configure WebLogic Tuxedo Connector and the Tuxedo Queuing Bridge (tBridge) using the WebLogic Server Administration Console. For additional information about the WebLogic Tuxedo Connector, see WebLogic Tuxedo Connector in the WebLogic Server documentation.

Log in to the WebLogic Server Administration Console. Perform the configuration steps in the order presented, using the instructions in the following sections.

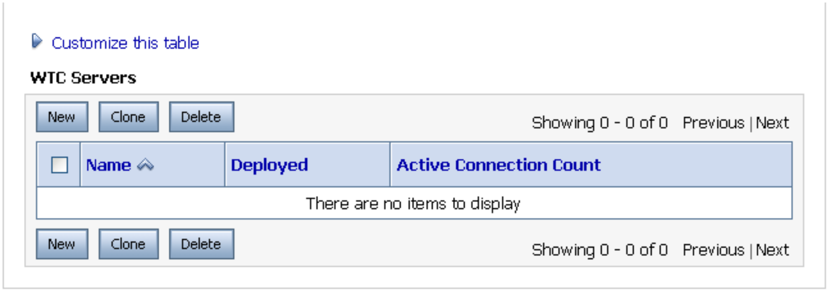
- Create a New WTC Server
- Create a Local Access Point
- Create a Remote Access Point
- Create a WTC Imported Service
- Create Field Table Classes (if Required)
- Create a Queuing Bridge
- Create a Redirection
- Activate Changes

Create a New WTC Server

Follow these steps:

1. Click **WTC Servers** under the Interoperability tab.
2. Click **Lock & Edit;** this allows you to make changes. A display similar to Figure 2-1 appears:

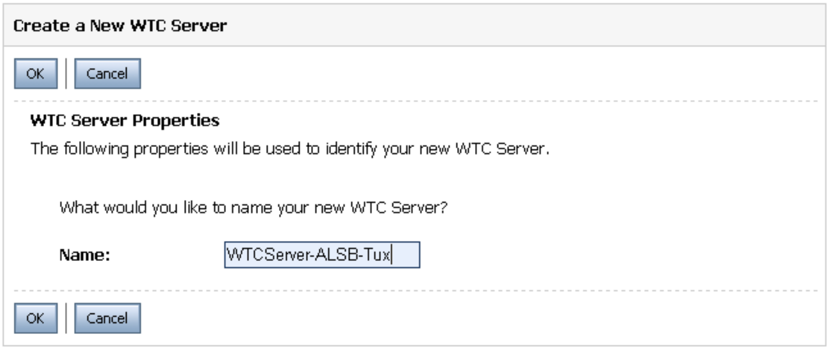
Figure 2-1 WTC Server Display



There are no items to display

3. Click **New** to add the new WTC server. A display similar to Figure 2-2 appears:

Figure 2-2 New WTC Server Data Entry Display



The following properties will be used to identify your new WTC Server.

What would you like to name your new WTC Server?

Name: WTCServer-ALSB-Tux

4. Enter a name for the WTC server and click **OK**.

   A message at the top of the page indicates that the server was added correctly.

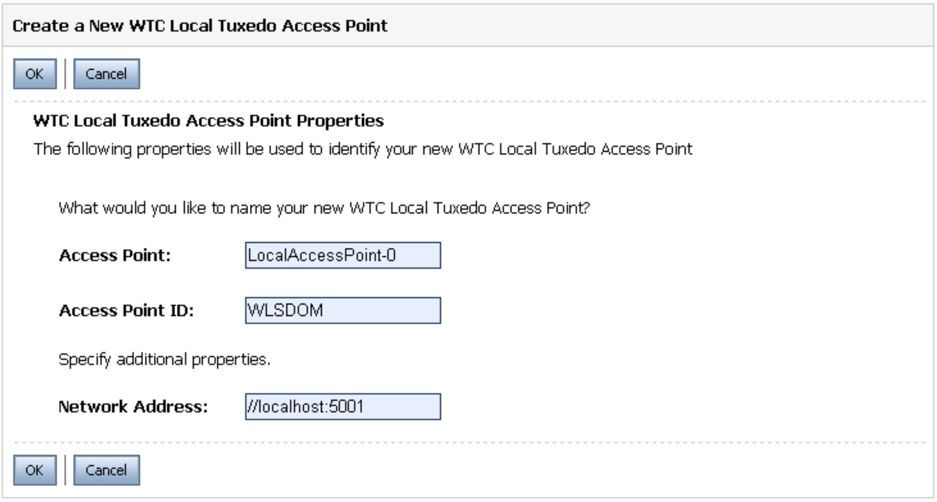
5. Click the newly created WTC server to display its settings.

Create a Local Access Point

Follow these steps:

1. Click **Local APs** on the Configuration tab.
2. Click **New** to create a new WTC Local Access Point. A display similar to Figure 2-3 appears:

Figure 2-3 New Local Access Point Data Entry Display



Create a New WTC Local Tuxedo Access Point

OK | Cancel

WTC Local Tuxedo Access Point Properties

The following properties will be used to identify your new WTC Local Tuxedo Access Point

What would you like to name your new WTC Local Tuxedo Access Point?

Access Point:	LocalAccessPoint-0
Access Point ID:	WLSDOM

Specify additional properties.

Network Address:	//localhost:5001

OK | Cancel

3. Enter the following values:

   Access Point – A name for this access point.

   Access Point ID – The name WebLogic Server will use to refer to the access point. This value must match the Remote Access Point ID that the Tuxedo domain gateway has been configured to use for this WTC instance.

   Network Address – This value must match the remote network address that the Tuxedo domain gateway has been configured to use for this WTC instance.

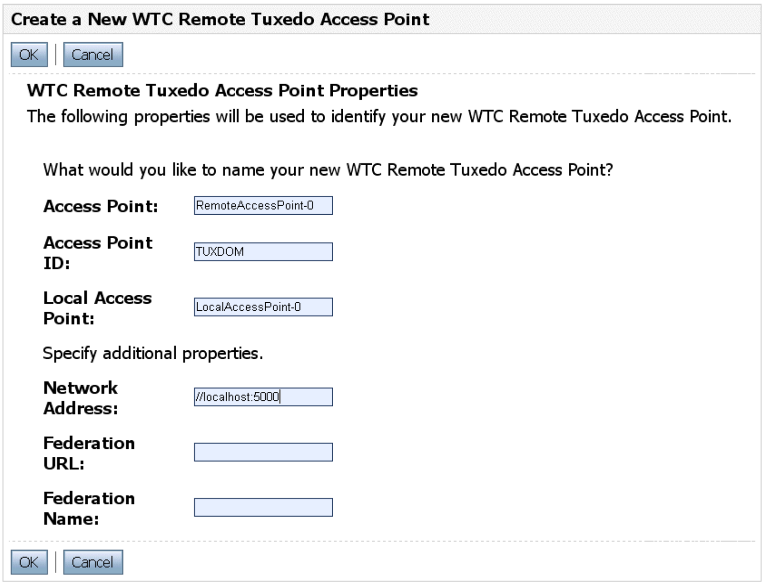
4. Click **OK**.

Create a Remote Access Point

Follow these steps:

1. Click **Remote APs** on the Configuration tab.
2. Click **New** to create a new WTC Remote Access Point. A display similar to Figure 2-4 appears:

Figure 2-4 New Remote Access Point Data Entry Display



WTC Remote Tuxedo Access Point Properties

The following properties will be used to identify your new WTC Remote Tuxedo Access Point.

What would you like to name your new WTC Remote Tuxedo Access Point?

Access Point:	RemoteAccessPoint-0
Access Point ID:	TUXDOM
Local Access Point:	LocalAccessPoint-0

Specify additional properties.

Network Address:	//localhost:5000
Federation URL:	
Federation Name:	

3. Enter the following values:

   Access Point – A name for this access point.

   Access Point ID – The name WebLogic Server will use to refer to the access point. This value must match the Local Access Point ID that the Tuxedo domain gateway has been configured to use for this WTC instance.

   Local Access Point – The name of WTC Local Access Point.

   Network Address – This value must match the local network address that the Tuxedo domain gateway has been configured to use for this WTC instance.

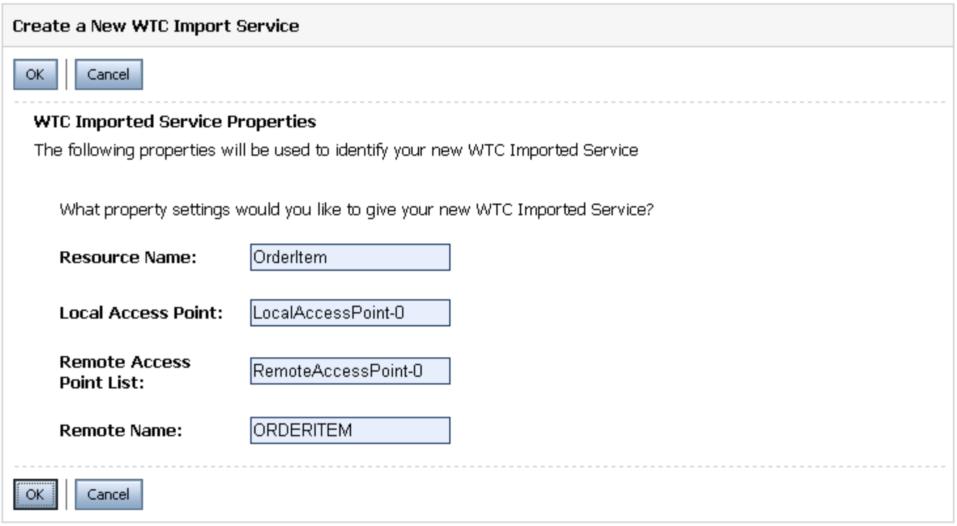
4. Click **OK**.

Create a WTC Imported Service

Follow these steps:

1. Click **Imported** on the Configuration tab.
2. Click **New** to create a new WTC Imported Service. A display similar to Figure 2-5 appears:

Figure 2-5 New WTC Import Service Data Entry Display


Create a New WTC Import Service

WTC Imported Service Properties

The following properties will be used to identify your new WTC Imported Service

What property settings would you like to give your new WTC Imported Service?

Resource Name:	OrderItem
Local Access Point:	LocalAccessPoint-0
Remote Access Point List:	RemoteAccessPoint-0
Remote Name:	ORDERITEM

3. Enter the following values:

   Resource Name – The name WebLogic Server will use to refer to the service (including the tBridge)

   Local Access Point – The name of the just created local access point

   Remote Access Point List – The name of the newly created remote access point

   Remote Name – The name of the service as exported by the remote Tuxedo system

4. Click **OK**.

Create Field Table Classes (if Required)

If the Tuxedo service expects FML or FML32 buffers, you must add one or more Field Table classes to a resource section in the WebLogic Tuxedo Connector configuration. To create the classes, use the weblogic.wtc.jatmi.mkfldclass utilities for FML field tables or the weblogic.wtc.jatmi.mkfldclass32 utility for FML32 field tables.

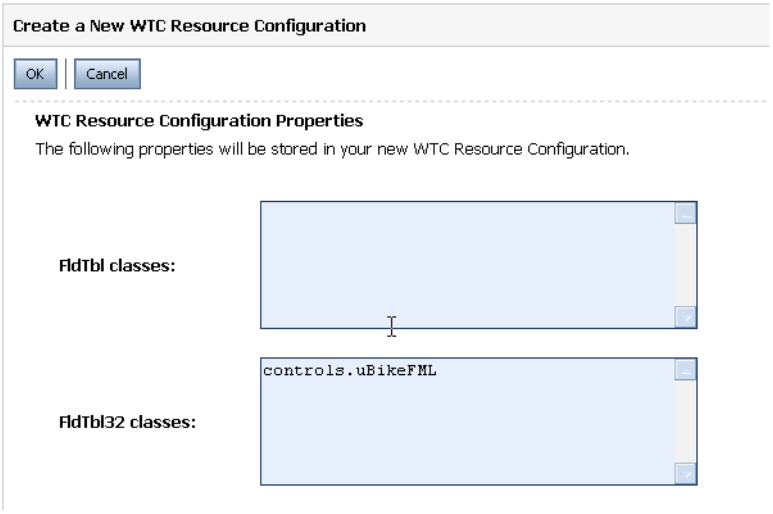
If you do not need to create Field Table classes, skip to "Create a Queuing Bridge" on page 2-9.

To create Field Table classes, follow these steps:

1. Click **Resources** on the Configuration tab.

2. Click **New** to create a new WTC Resource Configuration. A display similar to Figure 2-6 appears:

Figure 2-6 New Field Table Class Data Entry Display



WTC Resource Configuration Properties

The following properties will be stored in your new WTC Resource Configuration.

FldTbl classes:

FldTbl32 classes: controls.uBikeFML

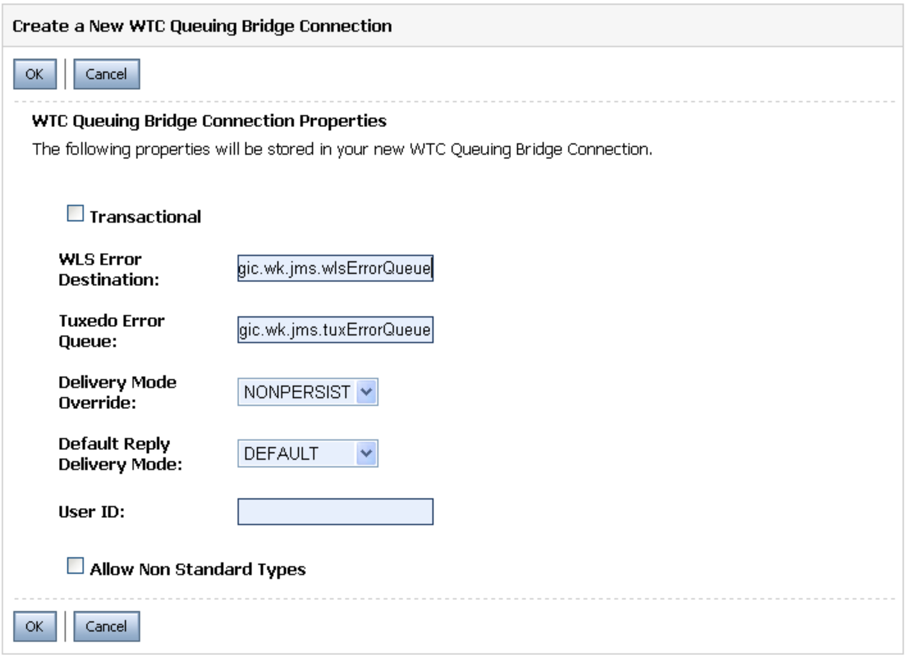
3. Add the full class names to the **FldTbl classes** or **FldTbl32 classes** fields. The classes must be on the WebLogic Server classpath; you might need to change the WebLogic Server classpath. For information on setting the WebLogic Server classpath, see Modifying the Classpath in the WebLogic Server *Command Reference*.

Create a Queuing Bridge

Follow these steps:

1. Click **Queuing Bridge** on the Configuration tab.
2. Click **New** to create a new Queuing Bridge. A display similar to Figure 2-7 appears:

Figure 2-7 New WTC Queuing Bridge Data Entry Display



Create a New WTC Queuing Bridge Connection

WTC Queuing Bridge Connection Properties

The following properties will be stored in your new WTC Queuing Bridge Connection.

☐ Transactional	
WLS Error Destination:	gic.wk.jms.wlsErrorQueue
Tuxedo Error Queue:	gic.wk.jms.tuxErrorQueue
Delivery Mode Override:	NONPERSIST
Default Reply Delivery Mode:	DEFAULT
User ID:	
☐ Allow Non Standard Types	

3. Enter the following values:

   WLS Error Destination – Enter the JNDI name of the JMS queue that should receive messages if they can't successfully be delivered to Tuxedo.

   Tuxedo Error Queue – Enter the JNDI name of the JMS queue that is to receive error replies from Tuxedo.

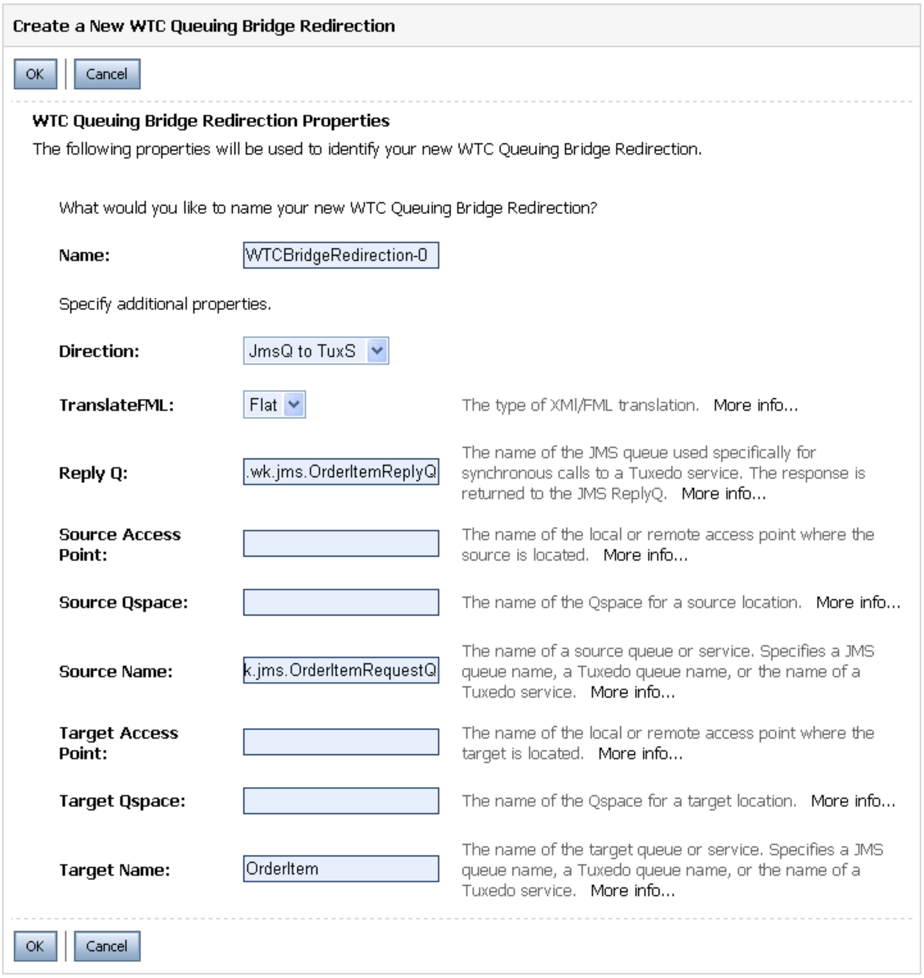
4. Click **OK**.

Create a Redirection

Follow these steps:

1. Click **Redirections** on the Configuration tab.
2. Click **New** to create a redirection for the service that is to be called in Tuxedo. A display similar to Figure 2-8 appears:

Figure 2-8 New WTC Redirection Data Entry Display



Create a New WTC Queuing Bridge Redirection

OK | Cancel

WTC Queuing Bridge Redirection Properties

The following properties will be used to identify your new WTC Queuing Bridge Redirection.

What would you like to name your new WTC Queuing Bridge Redirection?

Name: WTCBridgeRedirection-0

Specify additional properties.

Direction:	JmsQ to TuxS	
TranslateFML:	Flat	The type of XMI/FML translation. More info...
Reply Q:	.wk.jms.OrderItemReplyQ	The name of the JMS queue used specifically for synchronous calls to a Tuxedo service. The response is returned to the JMS ReplyQ. More info...
Source Access Point:		The name of the local or remote access point where the source is located. More info...
Source Qspace:		The name of the Qspace for a source location. More info...
Source Name:	k.jms.OrderItemRequestQ	The name of a source queue or service. Specifies a JMS queue name, a Tuxedo queue name, or the name of a Tuxedo service. More info...
Target Access Point:		The name of the local or remote access point where the target is located. More info...
Target Qspace:		The name of the Qspace for a target location. More info...
Target Name:	OrderItem	The name of the target queue or service. Specifies a JMS queue name, a Tuxedo queue name, or the name of a Tuxedo service. More info...

OK | Cancel

3. Enter the following values:

   Direction –Specify **JmsQ to TuxS**.

   TranslateFML – If the Tuxedo service requires XML-to-FML and FML-to-XML translation, select **Flat**.

   Reply Q – Enter the JNDI name of the JMS queue that is to receive the replies from the Tuxedo service.

Source Name – Enter the JNDI name of the JMS queue that is to receive the requests for the Tuxedo service.

Target Access Point – Enter the name of the remote access point that you created previously.

Target Name – Enter the name of the imported Tuxedo service that you created previously.

4. Click **OK**.

Activate Changes

To activate the changes you made, click **Activate Changes** on the WebLogic Server Administration Console.

Configuring a New Business Service

To utilize the Tuxedo service from AquaLogic Service Bus, you must configure a new Business Service in the AquaLogic Service Bus Console. For more information about Business Services, see Business Services in the AquaLogic Service Bus *Console Online Help*.

Log in to the AquaLogic Service Bus Console. Perform the configuration steps in the order presented, using the instructions in the following sections.

- Add a New Project
- Add a Business Service
- Add a Proxy Service
- Configure the Proxy Service

Add a New Project

Follow these steps:

1. Click **Create** to start a new console session.

   You must be in a session to edit resources.

2. Click **Project Explorer**.

3. Enter a name for the new project and click **Add Project**.

   A message at the top of the page indicates that the server was added correctly.

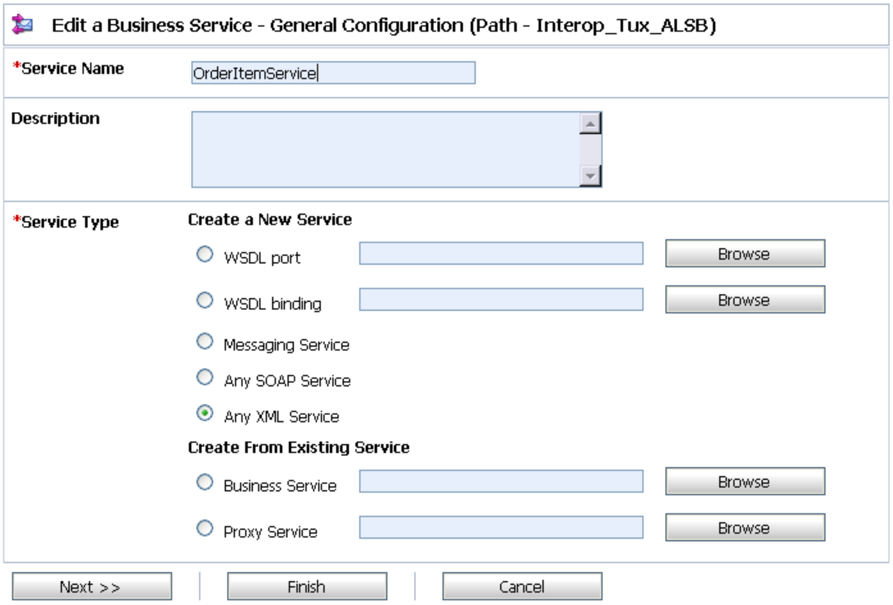
Add a Business Service

Follow these steps:

1. Click the newly created project.
2. In the **Resources** area **Create Resource** dropdown menu, select **Business Service**.

   The **Edit a Business Service – General Configuration** page displays, as shown in Figure 2-9.

Figure 2-9 New Business Service Page 1



Edit a Business Service - General Configuration (Path - Interop_Tux_ALSB)

*Service Name	OrderItemService
Description	
*Service Type	Create a New Service: ( ) WSDL port, ( ) WSDL binding, ( ) Messaging Service, ( ) Any SOAP Service, [✓ Any XML Service]; Create From Existing Service: ( ) Business Service, ( ) Proxy Service

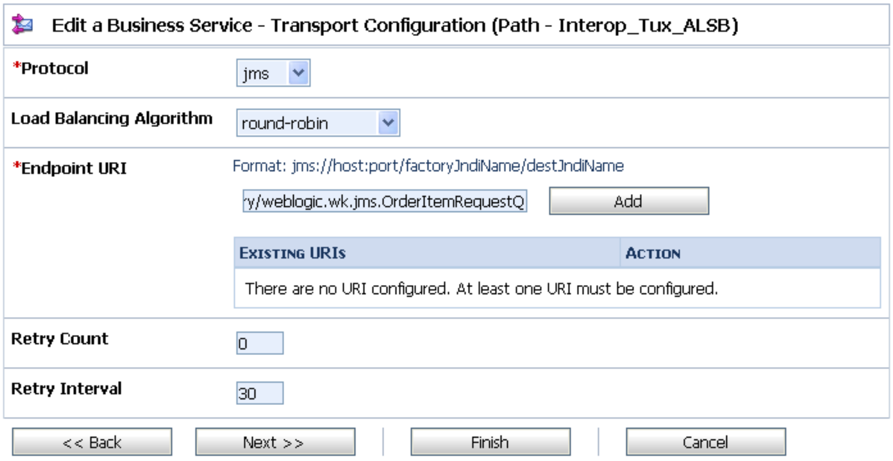
3. Enter the following values:

   Service Name – The name of the service

   Service Type – Select **Any XML Service** (the default)

   Click **Next** to display the **Edit a Business Service – Transport Configuration** page as shown in Figure 2-10.

Figure 2-10 New Business Service Page 2


Edit a Business Service - Transport Configuration (Path - Interop_Tux_ALSB)

*Protocol	jms
Load Balancing Algorithm	round-robin
*Endpoint URI	Format: jms://host:port/factoryJndiName/destJndiName y/weblogic.wk.jms.OrderItemRequestQ
Retry Count	0
Retry Interval	30

Existing URIs	Action
There are no URI configured. At least one URI must be configured.	

4. Enter the following values:

   Protocol – Select **jms.**

   Load Balancing Algorithm – Leave the default as is, or select another algorithm.

   Endpoint URI - Enter a JMS URI. that corresponds to the endpoint URI on the server where the service was deployed.

5. Click **Next** to continue.

6. Enter the following values:

   Is Response Required – Select the checkbox.

   Response URI – Enter a valid response URI.

   Response Timeout – 30.

   Message Type – **Text**.

7. Click **Finish**.

8. At the **Summary** page, click **Save**.

Add a Proxy Service

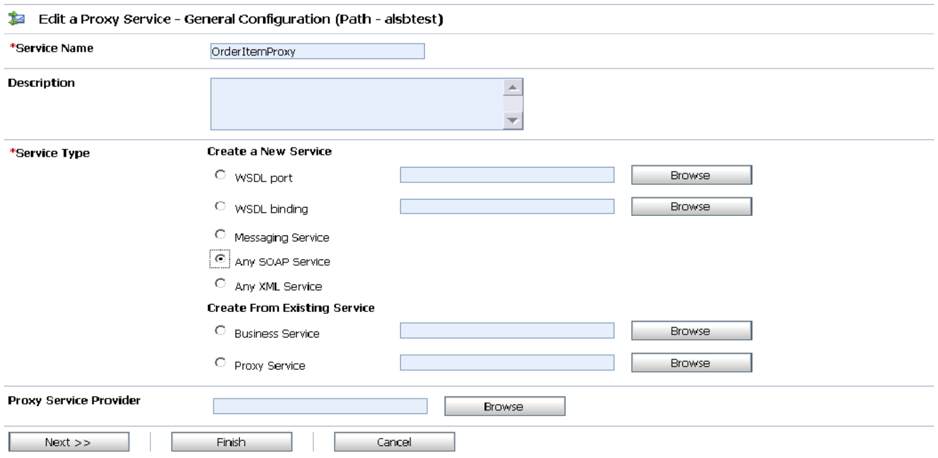
Create a proxy service for testing purposes. For more information about proxy services, see Proxy Services in the AquaLogic Service Bus *Console Online Help*.

Follow these steps:

1. In the **Resources** area **Create Resource** dropdown menu, select **Proxy Service**.

   The **Edit a Proxy Service – General Configuration** page displays, as shown in Figure 2-11.

Figure 2-11 New Proxy Service Data Entry Page 1



Edit a Proxy Service - General Configuration (Path - alsbtest)

*Service Name	OrderItemProxy
Description	
*Service Type	Create a New Service: ( ) WSDL port, ( ) WSDL binding, ( ) Messaging Service, [✓ Any SOAP Service], ( ) Any XML Service; Create From Existing Service: ( ) Business Service, ( ) Proxy Service
Proxy Service Provider	

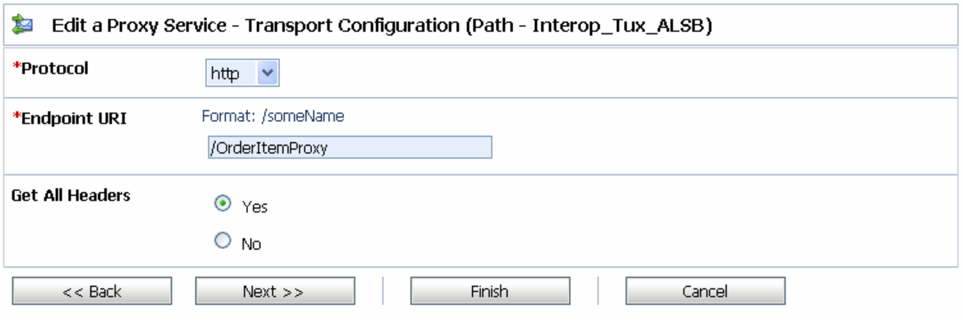
2. Enter the following values:

   Service Name – The name of the service

   Service Type – Select **Any SOAP Service**

   Click **Next** to display the **Edit a Proxy Service – Transport Configuration** page, as shown in Figure 2-12.

Figure 2-12 New Proxy Service Data Entry Page 2



Edit a Proxy Service - Transport Configuration (Path - Interop_Tux_ALSB)

*Protocol: http

*Endpoint URI: Format: /someName — /OrderItemProxy

Get All Headers: [✓ Yes] No

3. Enter the following required values:

   Protocol – Select **http**.

   Endpoint URI - The URI field automatically displays a URI corresponding to your proxy service name. You can leave it as is, or enter a different URI.

4. Click **Finish**.

5. At the Summary page, click **Save**.

Configure the Proxy Service

AquaLogic Service Bus Message Flows define the implementation of proxy services. Message flows can include zero or more pipeline pairs: request and response pipelines for the proxy service (or for the operations on the service) and error handler pipelines that can be defined for stages, pipelines, and proxy services. Pipelines can include one or more stages, which in turn include actions. To change the routing behavior of the proxy service you will edit this message flow to:

- Add a route node
- Configure an action to route the Proxy Service to the Business Service resource that you created previously

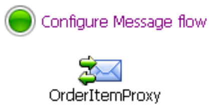
Follow these steps:

1. In the AquaLogic Service Bus Console navigation panel, select Resource Browser from the list of available choices, if it is not already selected.

   The Resource Browser pane is opened in the navigation panel and the Summary of Proxy Services project page is displayed in the console.

2. In Options, click the **Message Flow** icon . A display similar to Figure 2-13 appears:

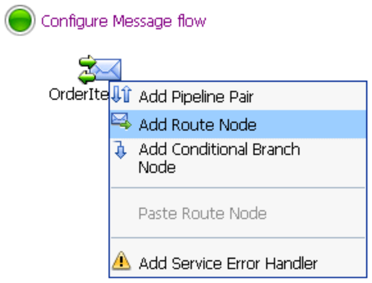
Figure 2-13 Message Flow Default Display



The Edit Message Flow page for the proxy service you created previously is displayed. This page displays the default message flow configuration. The default configuration consists of a start node. This is the minimum configuration of a message flow. The behavior of the message flow is sequential.

3. Click the Start Node. From the popup menu select the **Add Route Node** link, as shown in Figure 2-14.

Figure 2-14 Convert to Route Node Display

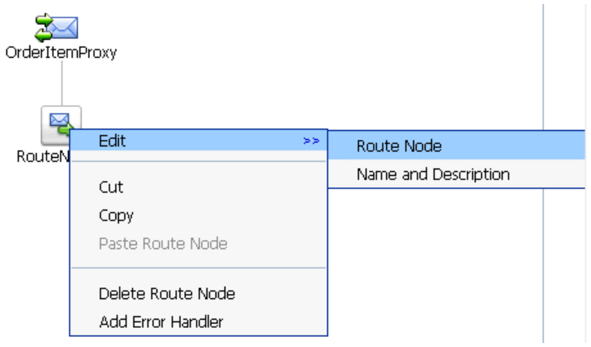


4. In the configuration dialog, name the route node as desired and click **Save**.

   In the message flow, the name of the node changes to display the route node name.

5. Click the route node and from the pop up menu select **Edit > Route Node**, as shown in Figure 2-15:

Figure 2-15 Edit Route Node Display

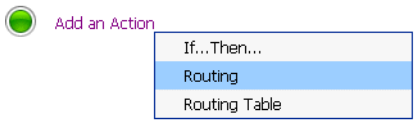


The Edit Stage Configuration page is displayed. The page contains a single link, Add an Action.

A stage is an element of a pipeline and it is a container for actions defined in a pipeline. Actions are the elements of a pipeline stage that define the handling of messages as they flow through a proxy service.

6. Click the **Add an Action** link, then select **Routing** from the popup menu, as shown in Figure 2-16:

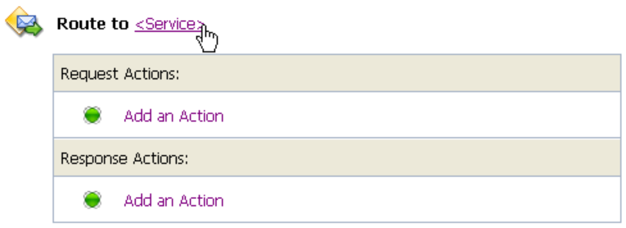
Figure 2-16 Message Flow Routing Display



The Edit Stage Configuration page changes to display the contents of the action. The contents of the action are defined by the type of node we created—a route node.

7. In Route to <Service>, click **<Service>**, as shown in Figure 2-17:

Figure 2-17 Route to Service Display



The Service Browser displays the names of the Proxy Service and Business Service that you created.

8. Select the Business Service that you created in "Add a Business Service" on page 2-13.
9. Click **Submit**.

   The display updates to show routing to the Business Service.

The configuration is completed and ready to test.

Testing Your Configuration

Now that you have configured AquaLogic Service Bus to work with BEA Tuxedo, you can test the application. One way to test the configuration is to use a WebLogic Workshop–based application, by setting up a web service proxy and calling it using a default pipeline.

The following list of tasks summarizes the process of testing outbound usage of BEA Tuxedo by AquaLogic Service Bus.

1. Build and start the Tuxedo servers and the WebLogic Workshop application.
2. Set up a WebLogic Workshop application to call the AquaLogic Service Bus proxy.
3. Run the web service in the WebLogic Workshop application, inputting a request. A successful response to the request indicates that the configuration is correct.

Using AquaLogic Service Bus Services from BEA Tuxedo (Inbound Example)

The following sections describe how to use AquaLogic Service Bus services from BEA Tuxedo:

- Implementation Overview
- Before You Begin
- Adding Field Classes to EJB's JAR File
- Testing Your Configuration

Implementation Overview

The WebLogic Tuxedo Connector allows BEA Tuxedo applications to call an EJB as though it were another BEA Tuxedo application. This scenario uses a particular example EJB that performs the function of receiving a Tuxedo service request, translating the buffer as necessary, and placing the resulting message on a JMS queue. The method described in this document is based on setting up an EJB using deployment descriptors; this method also provides translation of Tuxedo buffers to and from JMS messages.

Before You Begin

Gather the following information about the AquaLogic Service Bus application that BEA Tuxedo will use:

- ID of the Tuxedo local access point and add this ID as a WLS user.
- Network address of the Tuxedo local access point.
- Name of the exported Tuxedo service.
- Whether the service needs XML-to-FML and FML-to-XML conversion.

  If XML-to-FML and FML-to-XML conversion is needed, you will need to add one or more Field Table classes; these classes are bundled in the jar with the EJB and loaded dynamically from the EJB's jar.
- ID of the access point that the Tuxedo domain gateway will use to refer to this WebLogic Tuxedo Connector instance.
- Network address that the Tuxedo domain gateway has defined for this WebLogic Tuxedo Connector Local Access Point.

You must create the following JMS queues if they do not already exist:

- One or more queues on which to place requests to the AquaLogic Service Bus service
- A queue to pick up replies from the AquaLogic Service Bus service

Configuring the Sample EJB ToQSBean

Set environment entries in the `ejb-jar.xml` file and set the name of the exported service that WebLogic Tuxedo Connector will allow other domains to call. The following sections contain instructions for these tasks.

Edit the ejb-jar.xml File

In the `ejb-jar.xml` file, set the following environment entries:

- JMSConnectionFactory – Set this to the JNDI name of the JMS connection factory that the EJB should use.
- ToQueueName – Set this to the JNDI name of the JMS queue into which the EJB should place incoming requests. This string can contain the indicator “`%s,`” which causes the actual service name to be substituted. For example, if the EJB is deployed to provide the CREDITCHECK ATMI service, setting this string to `weblogic.jms.%sServiceRequestQ` results in a JNDI name of `weblogic.jms.CREDITCHECKServiceRequestQ`.
- ReplyQueueName – Set this to the JNDI name of the JMS queue into which AquaLogic Service Bus places replies. This string can contain a “`%s`” to make this setting more generic.
- FieldTables – A comma-separated list of the FML/FML32 field table classes required for any FML/XML translation.

Edit the weblogic-ejb-jar.xml File

In the `weblogic-ejb-jar.xml` file, set the EJB’s JNDI name to `tuxedo.services.`*`svcname`*`Home` where *`svcname`* is the ATMI service name that the deployment supports. This is the name of the exported service that WTC allows other domains to call.

For reference information on the structure of the `weblogic-ejb-jar.xml` file, see weblogic-ejb-jar.xml Deployment Descriptor Reference in *Programming WebLogic Enterprise JavaBeans*.

Adding Field Classes to EJB's JAR File

This task is optional if you set up field classes as described in "Create Field Table Classes (if Required)" on page 2-8. Any Field Table class files required for XML to FML and FML to XML conversion must be included in the EJB's JAR file. For instructions on performing this task, see the WebLogic Server *Javadoc*.

Building and Deploying the EJB

Deploying an EJB enables WebLogic Server to serve the components of an EJB to clients. You can deploy an EJB using one of several procedures, depending on your environment and whether or not your EJB is in production.

For general instructions on deploying WebLogic Server applications and modules, including EJBs, see *Deploying Applications to WebLogic Server* in the WebLogic Server documentation. For EJB-specific deployment issues and procedures, see Deployment Guidelines for Enterprise Java Beans in *Programming WebLogic Enterprise JavaBeans*.

Configuring WebLogic Tuxedo Connector

You configure WebLogic Tuxedo Connector using the WebLogic Server Administration Console. For additional information about the WebLogic Tuxedo Connector, see WebLogic Tuxedo Connector in the WebLogic Server documentation.

Log in to the WebLogic Server Administration Console and perform these steps in the order presented.

Note: If you performed the WTC setup described in "Configuring WebLogic Tuxedo Connector and the Tuxedo Queuing Bridge" on page 2-4, you can skip the corresponding tasks described here; the only required task is to create a WTC Export Service.

To complete this configuration, you will perform the tasks described in the following sections:

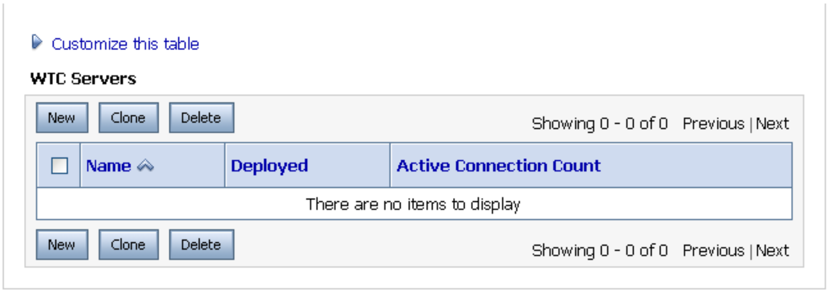
- Create a New WTC Server
- Create a Local Access Point
- Create a Remote Access Point
- Create a WTC Export Service

Create a New WTC Server

Follow these steps:

4. Click **WTC Servers** under the Interoperability tab.
5. Click **Lock & Edit;** this allows you to make changes. A display similar to Figure 2-18 appears:

Figure 2-18 WTC Server Display



WTC Servers

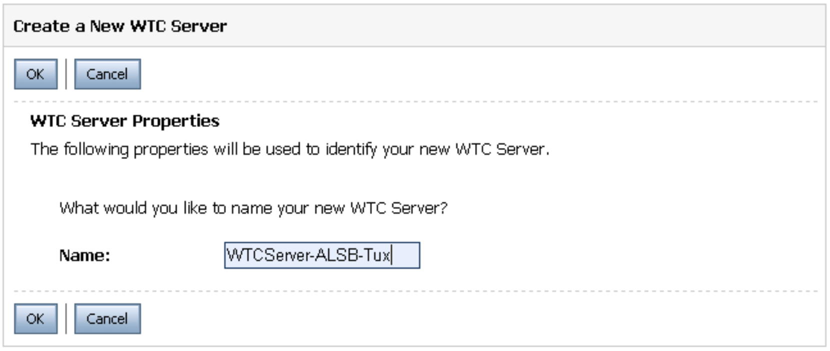
Showing 0 - 0 of 0

	Name	Deployed	Active Connection Count
There are no items to display			

Showing 0 - 0 of 0

6. Click **New** to add the new WTC server. A display similar to Figure 2-19 appears:

Figure 2-19 New WTC Server Data Entry Display



Create a New WTC Server

WTC Server Properties

The following properties will be used to identify your new WTC Server.

What would you like to name your new WTC Server?

Name: WTCServer-ALSB-Tux

7. Enter a name for the WTC server and click **OK**.

   A message at the top of the page indicates that the server was added correctly.

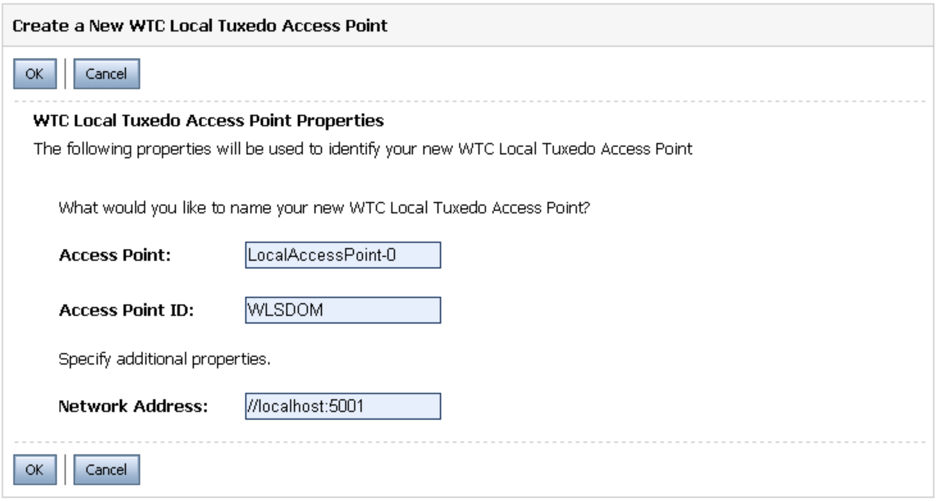
8. Click the newly created WTC server to display its settings.

Create a Local Access Point

Follow these steps:

1. Click **Local APs** on the Configuration tab.
2. Click **New** to create a new WTC Local Access Point. A display similar to Figure 2-20 appears:

Figure 2-20 New Local Access Point Data Entry Display


Create a New WTC Local Tuxedo Access Point

WTC Local Tuxedo Access Point Properties

The following properties will be used to identify your new WTC Local Tuxedo Access Point

What would you like to name your new WTC Local Tuxedo Access Point?

Access Point: LocalAccessPoint-0

Access Point ID: WLSDOM

Specify additional properties.

Network Address: //localhost:5001

3. Enter the following values:

   Access Point – A name for this access point.

   Access Point ID – The name WebLogic Server will use to refer to the access point. This value must match the Remote Access Point ID that the Tuxedo domain gateway has been configured to use for this WTC instance.

   Network Address – This value must match the remote network address that the Tuxedo domain gateway has been configured to use for this WTC instance.

4. Click **OK**.

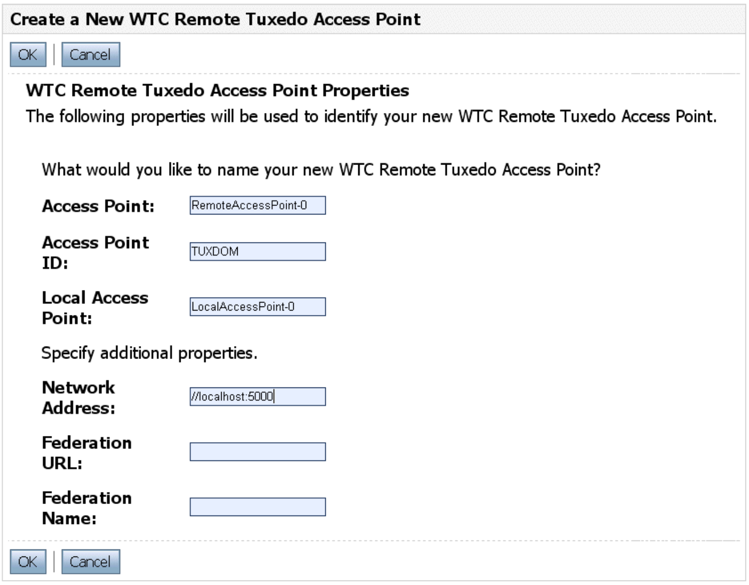
Create a Remote Access Point

Follow these steps:

1. Click **Remote APs** on the Configuration tab.

2. Click **New** to create a new WTC Remote Access Point. A display similar to Figure 2-21 appears:

Figure 2-21 New Remote Access Point Data Entry Display



Create a New WTC Remote Tuxedo Access Point

WTC Remote Tuxedo Access Point Properties

The following properties will be used to identify your new WTC Remote Tuxedo Access Point.

What would you like to name your new WTC Remote Tuxedo Access Point?

Access Point:	RemoteAccessPoint-0
Access Point ID:	TUXDOM
Local Access Point:	LocalAccessPoint-0

Specify additional properties.

Network Address:	//localhost:5000
Federation URL:	
Federation Name:	

3. Enter the following values:

   Access Point – A name for this access point.

   Access Point ID – The name WebLogic Server will use to refer to the access point. This value must match the Local Access Point ID that the Tuxedo domain gateway has been configured to use for this WTC instance.

   Local Access Point – The name of WTC Local Access Point.

   Network Address – This value must match the local network address that the Tuxedo domain gateway has been configured to use for this WTC instance.

4. Click **OK**.

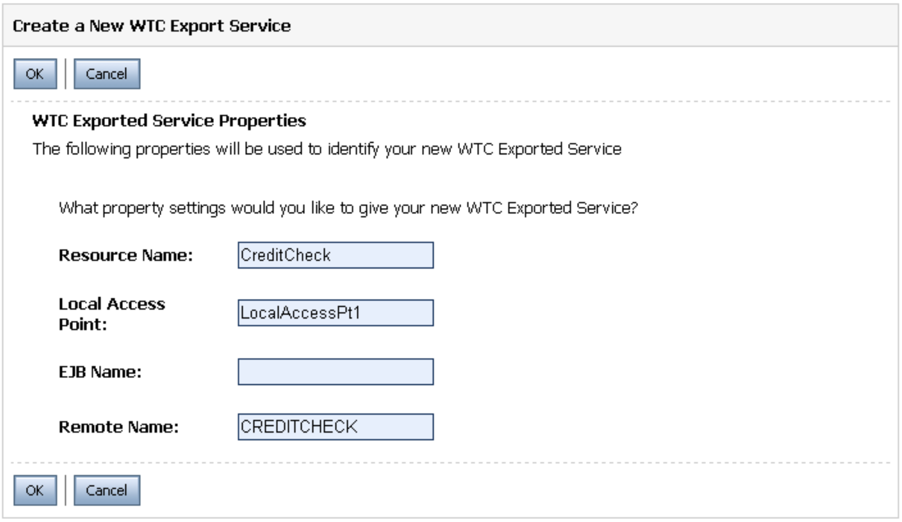
Create a WTC Export Service

Follow these steps:

1. Click **Exported** on the Configuration tab.

2. Click **New** to create a new WTC Exported Service. A display similar to Figure 2-22 appears:

Figure 2-22 New WTC Export Service Data Entry Display



WTC Exported Service Properties

The following properties will be used to identify your new WTC Exported Service

What property settings would you like to give your new WTC Exported Service?

Resource Name:	CreditCheck
Local Access Point:	LocalAccessPt1
EJB Name:	
Remote Name:	CREDITCHECK

3. Enter the following values:

   Resource Name – The name WebLogic Server will use to refer to the service (including the tBridge)

   Local Access Point – The name of the local access point you created earlier

   EJB Name – The complete name of the EJB home interface to use when invoking a service

   Remote Name – The name of the service as exported by the remote Tuxedo system

4. Click **OK**.

Adding and Configuring a Proxy Service

To utilize theAquaLogic Service Bus service from Tuxedo, you must configure a new proxy service using the AquaLogic Service Bus Console. For more information about proxy services, see Proxy Services in the AquaLogic Service Bus *Console Online Help*.

Log in to the AquaLogic Service Bus Console and perform these steps in the order presented.

To complete this configuration, you will perform the tasks described in the following sections:

- Add a New Project
- Add a Proxy Service
- Configure the Proxy Service

Add a New Project

Follow these steps:

1. Click **Create** to start a new console session.

   You must be in a session to edit resources.

2. Click **Project Explorer**.
3. Enter a name for the new project and click **Add Project**.

   A message at the top of the page indicates that the server was added correctly.

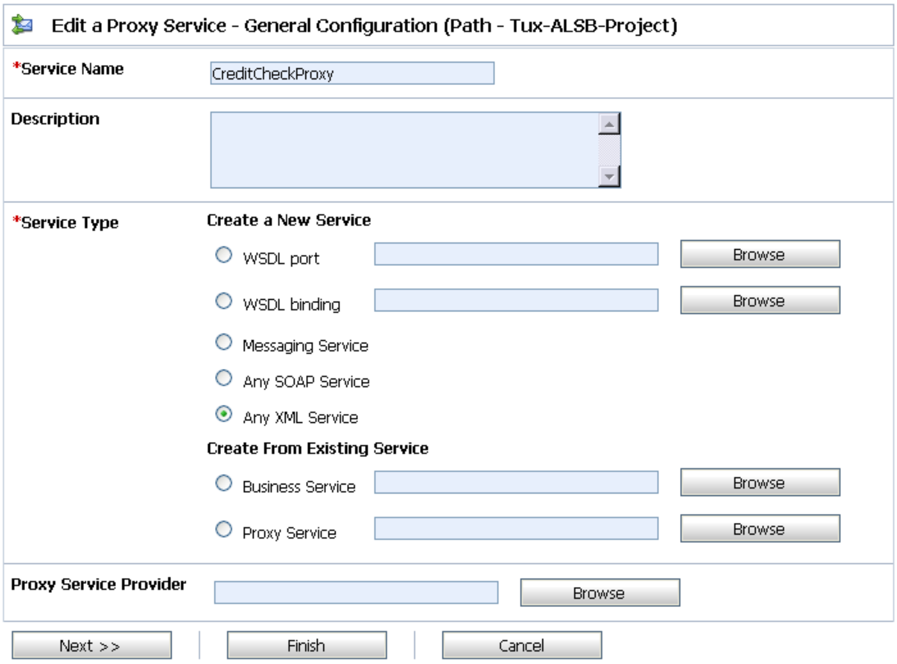
Add a Proxy Service

Follow these steps:

1. In the **Resources** area **Create Resource** dropdown menu, select **Proxy Service**.

   The **Edit a Proxy Service – General Configuration** page displays, as shown in Figure 2-23.

Figure 2-23 New Proxy Service Data Entry Page 1 - Inbound



Edit a Proxy Service - General Configuration (Path - Tux-ALSB-Project)

*Service Name	CreditCheckProxy
Description	
*Service Type	Create a New Service
	( ) WSDL port
	( ) WSDL binding
	( ) Messaging Service
	( ) Any SOAP Service
	[✓ Any XML Service]
	Create From Existing Service
	( ) Business Service
	( ) Proxy Service
Proxy Service Provider	

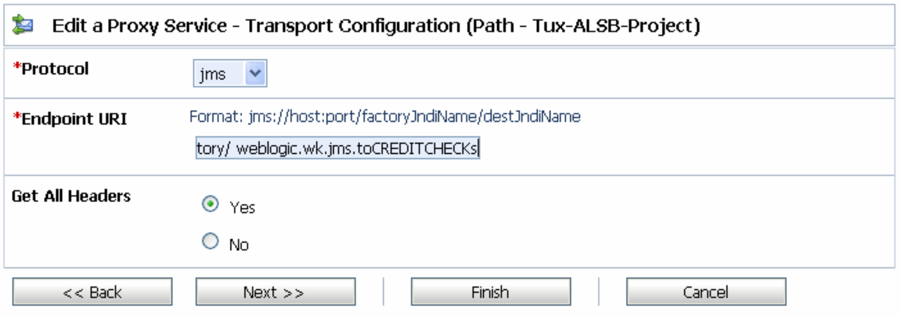
2. Enter the following values:

   Service Name – The name of the service

   Service Type – Select **Any XML Service** (the default)

   Click **Next** to display the **Edit a Proxy Service – Transport Configuration** page, as shown in Figure 2-24.

Figure 2-24 New Proxy Service Data Entry Page 2 - Inbound


Edit a Proxy Service - Transport Configuration (Path - Tux-ALSB-Project)

*Protocol: jms

*Endpoint URI: Format: jms://host:port/factoryJndiName/destJndiName

tory/ weblogic.wk.jms.toCREDITCHECKs

Get All Headers: [✓ Yes] No

3. Enter the following required values:

   Protocol – Select **jms**.

   Endpoint URI - Enter a JMS URI. that corresponds to the endpoint URI on the server where the service was deployed.

4. Click **Next** to continue.

5. Enter the following values:

   Is Response Required – Select the checkbox.

   Message Type – **Text**.

6. Click **Finish**.

7. At the **Summary** page, click **Save**.

Configure the Proxy Service

AquaLogic Service Bus Message Flows define the implementation of proxy services. Message flows can include zero or more pipeline pairs: request and response pipelines for the proxy service (or for the operations on the service); and error handler pipelines that can be defined for stages, pipelines, and proxy services. Pipelines can include one or more stages, which in turn include actions. To change the routing behavior of the proxy service you will edit this message flow to:

- Add a route node
- Configure an action to route the Proxy Service to the Business Service resource that you created previously

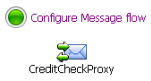
Follow these steps:

1. In the AquaLogic Service Bus Console navigation panel, select Resource Browser from the list of available choices, if it is not already selected.

   The Resource Browser pane is opened in the navigation panel and the Summary of Proxy Services project page is displayed in the console.

2. In Options, click the **Message Flow** icon . A display similar to Figure 2-25 appears:

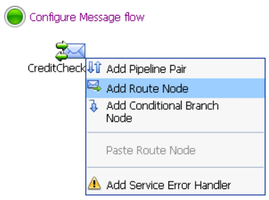
Figure 2-25 Message Flow Default Display



   The Edit Message Flow page for the proxy service you created previously is displayed. This page displays the default message flow configuration. The default configuration consists of a start node. This is the minimum configuration of a message flow. The behavior of the message flow is sequential.

3. Click the Start Node. From the popup menu select the **Add Route Node** link, as shown in Figure 2-26.

Figure 2-26 Convert to Route Node Display

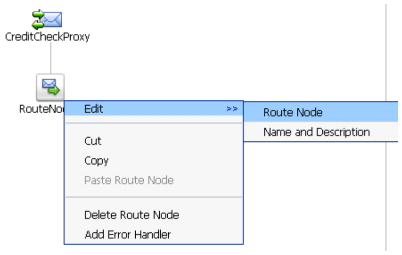


4. In the configuration dialog, name the route node as desired and click **Save**.

   In the message flow, the name of the node changes to display the route node name.

5. Click the route node and from the pop up menu select **Edit > Route Node**, as shown in Figure 2-27:

Figure 2-27 Edit Route Node Display

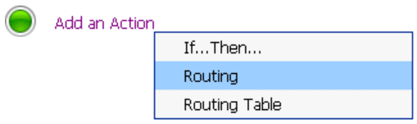


The Edit Stage Configuration page is displayed. The page contains a single link, Add an Action.

A stage is an element of a pipeline and it is a container for actions defined in a pipeline. Actions are the elements of a pipeline stage that define the handling of messages as they flow through a proxy service.

6. Click the **Add an Action** link, then select **Routing** from the popup menu, as shown in Figure 2-28:

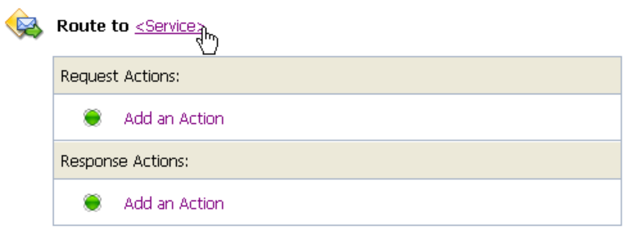
Figure 2-28 Message Flow Routing Display



The Edit Stage Configuration page changes to display the contents of the action. The contents of the action are defined by the type of node we created—a route node.

7. In Route to <Service>, click **<Service>**, as shown in Figure 2-29:

Figure 2-29 Route to Service Display



The Service Browser displays the names of the Proxy Service and Business Service that you created.

8. Select the Business Service that you want to expose to Tuxedo.
9. Click **Submit**.

   The display updates to show routing to the Business Service.

The configuration is completed and ready to test.

Testing Your Configuration

Now that you have configured BEA Tuxedo to work with AquaLogic Service Bus, you can perform a test to verify that it is working correctly. If you are using XML-to-FML32 and FML32-to-XML conversions, you can test this configuration using the "ud32" Tuxedo client program that is included with BEA Tuxedo. (If you are using FML conversions, you can use the "ud" client.) ud32 reads input consisting of text representation of FML buffers. For more information, see the information on the ud and ud32 commands in the *Tuxedo Command Reference*.

If you are not using XML-to-FML and FML-to-XML conversions, you must develop a test client program in Tuxedo to test this configuration. To find information on this task, refer to the BEA Tuxedo documentation.

Interoperability with JMS

AquaLogic Service Bus is certified against the following JMS implementations:

- WebLogic Server 9.x JMS
- IBM WebSphere MQ/JMS 5.3

Configuring proxy services and business services to use the JMS transport is described in the Proxy Services and Business Services sections of the *Using the AquaLogic Service Bus Console*.

All of the AquaLogic Service Bus service types support the JMS transport. For information about the AquaLogic Service Bus service types and the transports for each of the service types, see “Selecting a Service Type” in Modeling Message Flow in AquaLogic Service Bus in the *BEA AquaLogic Service Bus User Guide*.

For information about WebLogic Server 9.x JMS, see the following resources:

- Managing Your Applications in *Programming WebLogic JMS*
- Configure JMS Servers in the *WebLogic Server Administration Console Online Help*

This section includes the following additional JMS interoperability topics:

- Asynchronous Request/Response
- SOAP/JMS Transport
- Naming Guidelines for WebLogic Servers, JMS Servers, and Domains
- Specifying the JMS Type for Services

- AquaLogic Service Bus and MQ/JMS Interoperability
- WSDL-Defined SOAP Fault Messages

Asynchronous Request/Response

For information about designing asynchronous request/response messaging, including the use of the JMS Correlation ID to link the request and response messages, see "Asynchronous Request/Response" in Modeling Message Flow in AquaLogic Service Bus in the *BEA AquaLogic Service Bus User Guide*.

SOAP/JMS Transport

When using the JMS binding to configure a business service in BEA WebLogic Workshop, the SOAP/JMS URI format you must provide in the AquaLogic Service Bus Console is:

```
jms://host:port/factoryJndiName/destJndiName
```

However, BEA WebLogic Workshop expects the following format:

```
jms://host:port/factoryJndiName/destJndiName?URI=/process/myprocess.jpd
```

To overcome this problem, you must set the URI as a JMS property inside the message flow on the outbound variable (`$outbound`) before it is sent. For information about setting `$outbound`, see "Inbound and Outbound Variables" in Message Context in the *Using the AquaLogic Service Bus Console*.

When using the JMS binding to configure a business service in WebLogic Server 8.1, you must use the following SOAP/JMS URI format in the AquaLogic Service Bus Console:

```
jms://host:port/factoryJndiName/destJndiName?URI=/contextURI/serviceName
```

You must set the URI as a JMS property inside the message flow on the outbound variable (`$outbound`) before a request is sent to the business service.

When using the JMS binding to configure a business service in WebLogic Server 9.x, you must use the following SOAP/JMS URI format in the AquaLogic Service Bus Console:

```
jms://host:port/contextURI/serviceName?URI=destJndiName
```

You must configure the business service to use the JNDI name of an existing `QueueConnectionFactory` in the target WebLogic Server. You must also set a user defined JMS property with the name as URI and the value as `/contextURI/serviceName` inside the message flow on the outbound variable (`$outbound`) before a request is sent to the business service.

Naming Guidelines for WebLogic Servers, JMS Servers, and Domains

Unique naming rules apply to all WebLogic Server deployments if more than one domain is involved. Therefore, make sure of the following:

- WebLogic Server instances and domain names are unique.
- WebLogic JMS server names are uniquely named across domains.
- If a JMS file store is being used for persistent messages, the JMS file store name must be unique across domains.

Regarding JMS Server names:

- You cannot have duplicate JMS server names within the same domain. If you do, when messages are sent to a destination at a particular JMS server, it is ambiguous as to which server the message should be sent.
- If you are using Store and Forward (SAF), having duplicate JMS Server names in different domains does not pose a problem.
- In the case of cross-domain communication, having duplicate JMS Server names can be a problem when using the `ReplyTo` function. For a scenario in which two domains each have a JMS Server with the same name, a `ReplyTo` message sent from a given domain is returned to the JMS server on the same domain that received the message instead of being returned to the domain that sent the original message.

For more information about configuring and managing WebLogic JMS, see:

- Managing Your Applications in *Programming WebLogic JMS*
- Configure JMS Servers in the *WebLogic Server Administration Console Online Help*

For information about WebLogic Server Domains, see *Understanding Domain Configuration*.

Specifying the JMS Type for Services

To support interoperability with heterogeneous endpoints, AquaLogic Service Bus allows you to control the content type used, the JMS type used, and the encoding used when configuring message flows. The JMS type can be byte or text. For more information, see “Content Types, JMS Type, and Encoding” in Modeling Message Flow in AquaLogic Service Bus in the *BEA AquaLogic Service Bus User Guide*.

AquaLogic Service Bus and MQ/JMS Interoperability

For information about AquaLogic Service Bus and MQ/JMS interoperability, see Chapter 4, "Interoperability with WebSphere MQ."

WSDL-Defined SOAP Fault Messages

When consuming a WSDL that explicitly defines a fault, the WebLogic clientgen tool generates a subclass of `java.lang.Exception` for the XML fault type. When the WebLogic Server JAX-RPC stack inspects a SOAP response message and determines that the response message contains a SOAP fault, it tries to map the fault to a clientgen-generated exception Java class.

For example, if a WSDL contains the definitions shown in the following listing, the clientgen tool generates a Java class `com.bea.test.TheFaultType` that extends `java.lang.Exception`. A JAX-RPC client can catch `com.bea.test.TheFaultType` when invoking the related method of the service stub.

Listing 3-1 Example WSDL Definitions

```
<definitions ... xmlns:s0="http://www.bea.com/test/">
   ...
   <types>
      <xsd:schema targetNamespace="http://www.bea.com/test/">
         ...
         <xsd:complexType name="theFaultType">
            <xsd:sequence>
               <xsd:element name="ID" type="xsd:int" />
               <xsd:element name="message" type="xsd:string" />
            </xsd:sequence>
         </xsd:complexType>
         <xsd:element name="theFault" type="theFaultType" />
      </xsd:schema>
   </types>
```

```
      ...
      <message name="theFaultMessage">
         <part element="s0:theFaultPart" name="theFault" />
      </message>
      ...
      <binding ...>
         <operation ...>
            <soap:operation soapAction="..." style="document" />
            <input ...>
               ...
            </input>
            <output ...>
               ...
            </output>
            <fault ...>
               <soap:fault name="theFaultPart" use="literal" />
            </fault>
         </operation>
      </binding>
      ...
   </definitions>
```

The SOAP message must contain a fault of the correct format so that the JAX-RPC stack throws the correct exception. If the fault is constructed from inside a AquaLogic Service Bus message flow, you must:

1. Replace the node for the `$body` variable with the following example listing:

Listing 3-2

```
<soap-env:Body>
      <soap-env:Fault>
         <faultcode
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">soap:Server</fau
ltcode>
         <faultstring>Some literal string</faultstring>
         <detail>
            <test:theFault>
               <test:ID>Any user defined code</e2eb:Id>
               <test:message>A specific literal message</test:message>
            </test:theFault>
         </detail>
      </soap-env:Fault>
</soap-env:Body>
```

where:

– `soap-env` is the system prefix for the namespace `http://schemas.xmlsoap.org/soap/envelope/`

– `test` is the prefix for the namespace `http://www.bea.com/test/`.

  If the prefix `test` is not already known to AquaLogic Service Bus, you must declare it.

2. Configure a reply action with failure.

For information about configuring Reply Actions in the AquaLogic Service Bus Console, see Proxy Services Actions in *Using the AquaLogic Service Bus Console*.

The clientgen tool is used to generate the client-side artifacts, such as the JAX-RPC stubs, needed to invoke a Web Service. See Ant Task Reference in *Programming Web Services for WebLogic Server*.

Interoperability with WebSphere MQ

This section outlines how the AquaLogic Service Bus connects to WebSphere MQ and presents an overview of some of the message types used in communication between WebSphere MQ and AquaLogic Service Bus. Tuning guidelines for WebSphere MQ are also introduced.

The topics discussed in this section include:

Using WebSphere MQ in AquaLogic Service Bus

AquaLogic Service Bus connects to WebSphere MQ via the WebSphere MQ JMS interface. In other words, AquaLogic Service Bus is an implementation of the WebSphere MQ JMS Client. WebLogic Server's foreign JMS server specifies the initial context factory, connection factory, and queue to the WebSphere MQ server. WebSphere MQ JMS supports two transport types:

- BINDINGS
- CLIENT

If the WebSphere MQ JMS Client is running on the same physical machine as the queue manager, it is possible to set the transport type to BINDINGS. Otherwise, you can only use the CLIENT type.

WebSphere MQ can interface with AquaLogic Service Bus in two ways:

- AquaLogic Service Bus acts as the front-end of WebSphere MQ to accept service requests from other applications and translates them to WebSphere MQ requests. See Figure 4-1.
- WebSphere MQ sends messages to other applications via AquaLogic Service Bus. See Figure 4-2.

Figure 4-1 AquaLogic Service Bus Front End

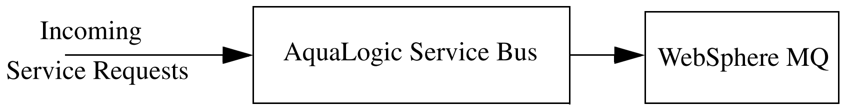
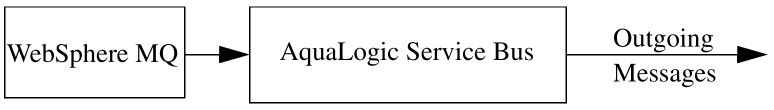


Figure 4-2 Messages sent via AquaLogic Service Bus



Configuration of AquaLogic Service Bus is performed in the AquaLogic Service Bus Console, which is described in the *AquaLogic Service Bus Console Online Help*.

Messaging Types

This section provides an overview of the following messaging types:

- Non-Persistent Messaging
- Non-XA Persistent Messaging
- XA Messaging

Non-Persistent Messaging

If you decide to accept unreliable delivery, such as some missing requests, you can use non-persistent messages where appropriate. WebSphere MQ logging and WebLogic JMS message persistence is only performed for persistent messages, therefore the use of non-persistent messages eliminates any related I/O activity.

Note: Non-persistent message throughput is usually limited by the processor speed of the machine. However, in case of a shortage of physical memory, the server system may consume CPU cycles on paging I/O.

Non-XA Persistent Messaging

WebSphere MQ persistent message throughput is usually limited by the queue manager and the I/O latency writing to the log.

XA Messaging

XA only applies to JMS to WebSphere MQ and WebSphere MQ to JMS messaging for AquaLogic Service Bus supported protocols. To enable queue support, the queue manager must be accessed using BINDINGS (AquaLogic Service Bus co-resides with the queue manager) or using CLIENT and a specific XA enabled WebSphere MQ client (for example, WebSphere MQ Extended Transactional Client) must be installed on the AquaLogic Service Bus machine.

Tuning WebSphere MQ

This section presents tips for tuning WebSphere MQ when working with AquaLogic Service Bus. For detailed WebSphere MQ information, see your relevant WebSphere MQ documentation.

- Use the BINDINGS transport type if AquaLogic Service Bus and the queue manager are deployed on the same machine.
- If you need XA for only a small section of application requests, create a separate connection object without XA enabled.
- Distribute active logs across many volumes. If your system must handle high persistent message throughput, you must place the log files on a fast Direct Access Storage Device (DASD) with a minimum of contention from other data set usage. Ideally, you should allocate each of the active logs on separate, low usage volumes.
- To reduce buffer overflow, tune the buffer pools and pagesets. Buffer overflow results in the flushing of the hard disk.

Interoperability with WebLogic Platform

AquaLogic Service Bus Interoperability with WebLogic 8.1 Domains

When your scenario requires interoperability between an AquaLogic Service Bus domain and a BEA WebLogic Platform 8.1 domain, you must set up your AquaLogic Service Bus domain as follows:

1. Apply any patches required for WebLogic Platform 8.1 domains. For the latest information about patches, see "Known Limitations" in the *BEA AquaLogic Service Bus Release Notes*.
2. Establish domain trust between all WebLogic Server domains participating in the transaction.

   For information about how to do so, see "Enabling Trust Between WebLogic Server Domains" in Configuring Security for a WebLogic Domain in *Securing WebLogic Server*.
3. Ensure that you set the **Security Interoperability Mode** attribute in the WebLogic Server Administration Console to **compatibility**.

   The **Security Interoperability Mode** attribute specifies the security mode to use for XA calls in cross-domain transactions. You can set this attribute on the WebLogic Server Administration Console from either the domain-wide security settings or from the Java Transaction API (JTA) configuration page for the WebLogic Server domain. For more information, see Domain: Security: General and Domains: Configuration: JTA in the *WebLogic Server Administration Console Online Help*.

4. Use different listening ports for the AquaLogic Service Bus and WebLogic 8.1 domains. Also, ensure that the Debug and Pointbase ports are different (you can check the `db.properties` file to determine which ports are specified). Both AquaLogic Service Bus and WebLogic 8.1 domains are initally set up with default ports, which may conflict if they not changed.

Guidelines for Naming WebLogic Domains and Servers

For information about naming WebLogic domains and servers, see "Naming Guidelines for WebLogic Servers, JMS Servers, and Domains" on page 3-3.

Related Topics

For AquaLogic Service Bus use cases, including use cases that involve interoperability between WebLogic Domains, see Modeling Message Flow in AquaLogic Service Bus in the *BEA AquaLogic Service Bus User Guide*.

Interoperability with Web Services for Remote Portlets (WSRP)

Web Services for Remote Portlets (WSRP) is an increasingly popular mechanism for generating markup fragments on a remote system for display in a local portal application. This section describes how AquaLogic Service Bus can be used to provide Service Level Agreement monitoring in applications that use WSRP.

The topics discussed in this section include:

The AquaLogic Service Bus Console, which is described in the *AquaLogic Service Bus Console Online Help*, is used to configure AquaLogic Service Bus. For more information about creating WSRP-enabled portals using WebLogic Portal, see *Using WSRP with WebLogic Portal*.

WSRP Producers and Consumers

WSRP involves two integral components:

- The remote application, called a *WSRP producer* (referred to as a *producer* in this section) implements standards-based Web Services using the SOAP specification over HTTP.

Producers can be easily created using WebLogic Portal or third-party implementations of WSRP.

- A *WSRP consumer* (referred to as a *consumer* in this section) is a Portal application. Typically, the consumer application references the producer's WSDL when the portal is designed, and the consumer directly accesses the producer.

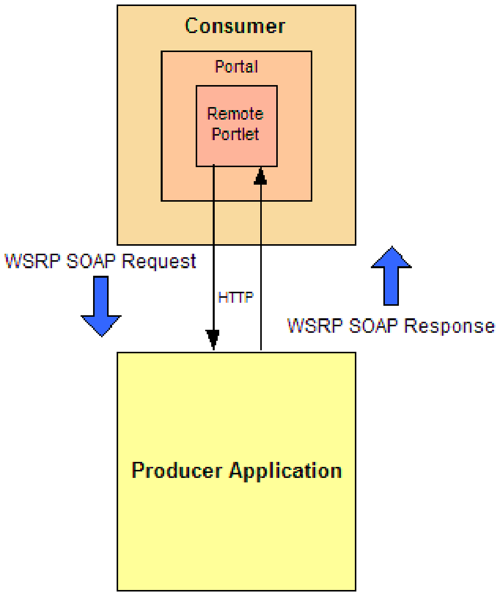
Architecture

This section describes the basic WSRP architecture and then shows how this architecture can be enhanced by adding AquaLogic Service Bus.

Basic WSRP Architecture

The following figure shows the basic WSRP SOAP request and response flow between a producer application and a consumer application.

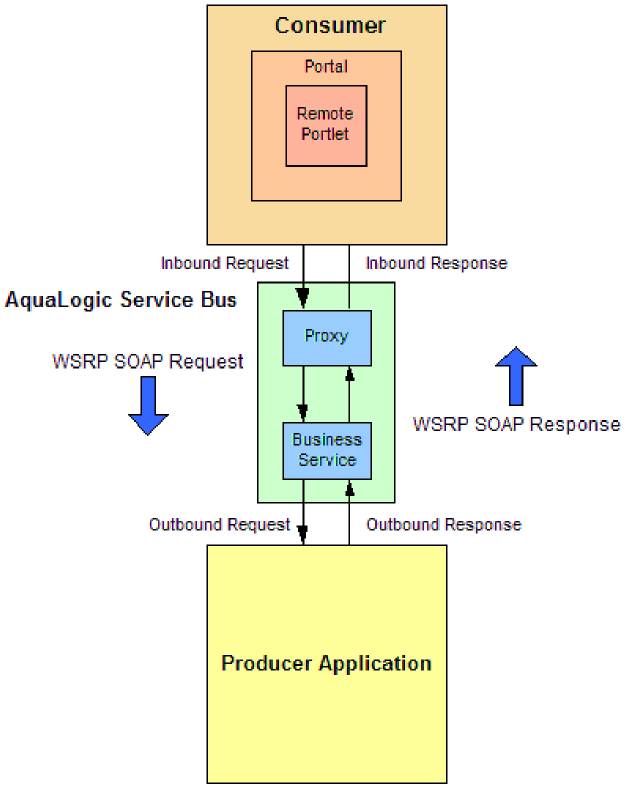
Figure 6-1 Basic Request/Response Flow Between Producer and Consumer Applications



Enhanced WSRP Architecture with AquaLogic Service Bus

Because a WSRP producer implements SOAP Web Services, an enterprise service bus (such as the AquaLogic Service Bus) can be used as an intermediary between the producer and consumer to provide Service Level Agreement monitoring, as shown in the following figure.

Figure 6-2 Enhanced WSRP Request / Response Flow Via AquaLogic Service Bus



In this architecture, the WSRP SOAP request / response flow occurs in the following sequence:

1. **Inbound Request:** The client (consumer) calls the proxy in the AquaLogic Service Bus.
2. **Outbound Request:** The proxy routes the request (a message containing the SOAP body and transport headers) to the business service, and then the business service makes the request of the external Web Service (producer).
3. **Inbound Response:** The Web service returns a reply to AquaLogic Service Bus.

4. **Outbound Response:** The proxy returns the reply (a message containing the SOAP body and transport headers) to the consumer.

The remainder of this section provides instructions for configuring the AquaLogic Service Bus to proxy service requests for WSRP services. It describes services that a producer provides, along with other attributes of WSRP that must be used to properly configure AquaLogic Service Bus. It provides different possible strategies that can be used to monitor producers with increasing degrees of detail. Finally, it discusses load balancing and failover with WSRP.

WSRP Design Concepts

This topic describes the following WSRP design concepts:

- WSRP WSDLs
- WSRP Messages

WSRP WSDLs

The following table describes the kinds of services offered by producers.

Table 6-1 Producer Services

Service	Description
Service Description	Required service. Used to describe the producer and the portlets that it makes available to consumers.
Markup	Required service. Manages user interaction with a remote portlet and returns the HTML markup used to render the portlet.
Registration	Optional service. Allows consumers to register themselves with the producer. Registration is required for complex producers.
Management	Optional service. Provided by complex producers for managing portlet customization and portlet preferences.
Markup Extension	Service provided by BEA Portal producers that replaces the Markup service. The Markup Extension allows more efficient message handling by using multipart MIME messages for transmitting HTML markup content.

Each producer implements a minimum of two services (Service Description and Markup). A *simple producer* offers just these two services. A *complex producer*, however, provides two additional services (Registration and Management). WebLogic Portal producers also implement an extension service (Markup Extension) that replaces the standard Markup service.

These services are described using a standard WSDL format. The producer supplies a single URL for retrieving its WSDL, which describes all of the services that are available from that producer. The endpoints for each service indicate whether the consumer should use transport-level security (HTTPS) or not to communicate with the producer.

WSRP Messages

WSRP uses SOAP over HTTP for all messages sent between producers and consumers. In addition to using standard message formats in the SOAP Body, WSRP requires that certain transport headers be set in the request message—at a minimum, consumers must set the `SOAPAction` header, cookie headers, and the usual HTTP headers (such as `Content-Type`). Producers will return a session cookie, plus any application-specific cookies, in the HTTP transport header of the response message. The consumer must return the session cookie in subsequent request messages.

Configuring AquaLogic Service Bus for WSRP

Configuring AquaLogic Service Bus for WSRP involves the following tasks:

- implementing a service that consumers can invoke to obtain an appropriate WSDL for a particular producer
- implementing the nuts and bolts of conveying a consumer's request to the producer and returning the response to the consumer

This topic describes the following tasks:

- Getting the Producer WSDL
- Routing Messages Between the Consumer and Producer
- Choosing the Monitoring Level

Getting the Producer WSDL

As a common practice, consumers contact a producer directly to obtain its WSDL. However, if AquaLogic Service Bus is used to proxy the service, then all access to the producer occurs via AquaLogic Service Bus. Therefore, a proxy service must be implemented for consumers that calls the producer's real URL to obtain its WSDL, and then transforms the results by:

- rewriting the endpoint address for the producer to refer to the Service Bus IP address and port
- changing the endpoint URI to refer to the AquaLogic Service Bus proxy service that reflects the required monitoring granularity (as described in "Choosing the Monitoring Level" on page 6-7)
- changing the endpoint protocol and port to reflect whether transport security is used between the consumer and the AquaLogic Service Bus proxy service

The developer who creates a producer can specify whether the producer requires SSL or not (`"secure=true"`). In addition, the AquaLogic Service Bus administrator can change the security requirement to the consumer via AquaLogic Service Bus configuration. For example, suppose a producer does not require SSL. The AquaLogic Service Bus administrator can require consumers to use SSL by:

- changing the WSDL to specify HTTPS
- configuring the proxy services for WSRP to use the HTTPS transport

When configured in this way, AquaLogic Service Bus automatically bridges the secure messages from the consumer to the non-secure messages used by the producer.

Routing Messages Between the Consumer and Producer

After the consumer has retrieved a copy of the WSDL, it uses the definitions in the WSDL to formulate service requests that it then sends to the producer via AquaLogic Service Bus. The WSRP request / response process involves the following steps:

1. The consumer sends a message to the AquaLogic Service Bus proxy service corresponding to the producer service.
2. The proxy service executes a simple message flow that routes the message (unchanged) to the actual producer service.
3. The producer formulates a response that it then sends to the consumer via AquaLogic Service Bus.

4. The consumer receives the response (unchanged) from the producer.

WSRP Web services expose portlets and those can rely on HTTP cookies and sessions. Therefore, WLSB must be configured to propagate HTTP transport headers (such as `SOAPAction` and cookies). However, by default, AquaLogic Service Bus does not pass transport headers from the proxy service to the business service, because it cannot assume that the proxy service uses the same transport as the business service. Therefore, the message flow must be configured to copy the request headers from the inbound request to the outbound request. Similarly, the response headers from the business service must be copied back to the proxy service's response to the consumer.

Although it is possible to copy *all* transport headers between the proxy service and the business service, it is necessary to be more selective to avoid errors. The Set-Cookie and Cookie headers must be copied. Because AquaLogic Service Bus is the entity that assembles the final message to send, it must own some of the headers information (such as `Content-Length`). For example, if the message flow were to copy the `Content-Length` header from the proxy service to the business service, it might result in an error because the length of the message could change during processing.

Choosing the Monitoring Level

When monitoring WSRP applications, an AquaLogic Service Bus administrator must decide about the degree of granularity that is required.

Table 6-2 WSRP Monitoring Levels

Monitoring Level	Description
Producer-level	Coarsest-grained level and the easiest to implement. This level looks at the producer as a whole, without regard to its constituent services.
Operation-level	Finest-grained monitoring level. Monitors the usage of a producer’s individual services and operations.

The decision about which monitoring level to implement has an impact on the complexity of the AquaLogic Service Bus configuration. It determines the type and number of proxies or business services that must be created for each producer. In addition, the AquaLogic Service Bus administrator can choose to monitor both the proxy service and the producer service—the granularity of monitoring does not need to be the same for each side.

Producer-Level Monitoring

Producer-level monitoring tracks the total number of requests sent to a producer, without regard to the specific service being requested. As such, producer-level monitoring is the simplest to configure within AquaLogic Service Bus. Because the service types are not significant, it is not necessary to create the proxy service or business service based on a WSDL. Instead, the service type is configured as `"Any SOAP Service"`. Each producer requires only a single proxy service and a single business service. For an example implementation, see "Producer-Level Monitoring Example" on page 6-12.

To configure producer-level monitoring, complete the following tasks:

1. Configure the message flow in the proxy service to unconditionally route any message to the business service.
2. Add a request action in the message flow to copy the appropriate request headers from the inbound request to the outbound request.
3. Add a response action in the message flow to copy the response headers back from the outbound response to the inbound response.

The suitability of producer-level monitoring depends on the specific requirements of a given implementation. In producer-level monitoring, the elapsed time for all services and operations for the producer are averaged together, regardless of the differences among them. However, a producer's services and operations can have vastly different characteristics, and it might not be meaningful to consider aggregated measurements. For example, the Markup service is the workhorse of WSRP—it requires substantially more time to execute than the Registration service. However, producer-level monitoring does not distinguish between the two. Nonetheless, producer-level monitoring can be useful to gauge the extent to which a producer is being utilized, or to help when there is a severe performance problem at the producer. Because the Markup service typically gets used more often (almost 99%) in a production system, it might still be useful to monitor Service Level Agreement (SLAs) at the producer level.

Operation-Level Monitoring

Operation-level monitoring tracks operations for services individually. Monitoring proxy services via operation-level monitoring is very easy to set up. Configuring operation-level monitoring for business services, however, requires more work. Fortunately, the message flow for WSRP services introduces very little overhead, and the mapping between proxy services and producers, and between business services and producers, is simple to configure. Therefore, to satisfy SLA requirements, it is often sufficient to monitor only the proxy services at the operation level. For an example implementation, see "Operation-Level Monitoring Example" on page 6-21.

Operation-Level Monitoring for Proxy Services

To configure operation-level monitoring for WSRP proxy services, create a proxy service for each of the services implemented by the producer.

- Simple producers require only two proxies—one for the Markup service and one for the Description service.
- Complex producers require these two proxies plus two additional proxy services for Registration and Management.

These proxy services should be based on the standard WSRP WSDLs using SOAP bindings. Only a single business service for the producer should be created, and it should be configured to use `"Any SOAP Service"` instead of being based on a WSDL. The message flow between the proxies and the business service should not modify the SOAP body in any way. However, just as for all WSRP message flows, it must pass the request headers via HTTP from the client request to the actual producer. Similarly, the response HTTP headers returned by the producer must be copied back to the client in the message flow.

Operation-Level Monitoring for Business Services

If operation-level monitoring is required for producer business services, then individual business services must be created for each of the Web services described in the producer's WSDL, and the business services must be defined using the WSDL. There is a one-to-one mapping between the proxy services and the business services—a simple, unconditional routing node is sufficient in the message flow.

For the operations to be counted correctly, AquaLogic Service Bus must be told which operation to use. Normally, the administrator would do this by selecting one of the operations from a drop-down menu when the business service is selected for the Route action. However, the operation specified by the client message is not the same for all messages, so a single, hard-coded value will not work here. The administrator must ensure that the business service uses the same operation as the proxy service. While this could be achieved by specifying a Routing Table action that selects the case using the `$operation` variable, it is a very tedious approach because the WSRP standard defines 14 operations across all WSRP services, and each would require a Route action with transformations to propagate the transport headers.

Fortunately, there is a more effective alternative. When routing to the business service, rather than selecting the operation from the drop-down menu, an administrator should use another transformation in the request actions to insert the value of `$inbound/ctx:service/ctx:operation` into `$outbound/ctx:service`. With this transformation, the operation for the business service is dynamically set to the same value as was specified for the proxy service, and AquaLogic Service Bus will correctly count and monitor all operations of the service.

Load Balancing and Failover

AquaLogic Service Bus allows business services to define multiple endpoints that all provide the same Web service. When multiple endpoints are defined, AquaLogic Service Bus can automatically load balance requests across endpoints, and it can automatically fail over requests when an endpoint is inaccessible. However, WSRP imposes some limitations on the use of these features.

Portlets are a means of surfacing a user interface to some application. Therefore, portlets typically have session data associated with them. To preserve session data, requests to the portlet must be directed to the same server (or cluster) that serviced the original request. This requirement makes load balancing via AquaLogic Service Bus inappropriate. Multiple endpoints in a business service will usually target different servers or cluster. Because there is no communication among servers that are in separate clusters, there is no way to preserve the session. Therefore, if multiple endpoints are defined for a WSRP business service, then the load balancing algorithm must be set to `"none"`.

Multiple endpoints can be used to provide redundancy in certain circumstances in the event that one of the endpoints is unavailable. The WSRP service is still available via a secondary endpoint. However, any session data that existed at the time the first endpoint failed will not be available on other endpoints.

This failover configuration is an option only for simple producers (see "WSRP WSDLs" on page 6-4), not for complex produces. Complex producers require that their consumers first register with the producer before sending service requests. The producer returns a registration handle that the consumer must include with each request to that producer. In the case where a business service defines multiple endpoints, each endpoint requires its own registration handle.

AquaLogic Service Bus is, however, stateless across requests—it does not maintain a mapping of the correct handle to send to a particular endpoint. In fact, it would only send the registration request to a single endpoint, so the consumer would be registered with only that one producer. If that one producer crashed, then AquaLogic Service Bus would route a service request to another endpoint defined for that business service, but the consumer would never have registered with that new producer, and the request would fail with an `"InvalidRegistration"` fault.

The management of registration handles therefore requires an application outside of AquaLogic Service Bus to maintain this state data. Error handling could be challenging to implement. Therefore, the registration requirement precludes defining multiple endpoints for complex producers. Because simple producers do not require or support the Registration service, a failover configuration that defines multiple endpoints in the business service is possible, although session data is lost on failover.

WSRP Interoperability Example

This section describes a WSRP interoperability example. It contains the following topics:

- Example Prerequisites
- Example Projects and Folders
- Producer-Level Monitoring Example
- Operation-Level Monitoring Example

Example Prerequisites

The WSRP interoperability example assumes the following components and configuration:

- WebLogic Platform 8.1 SP4
- AquaLogic Service Bus 2.0
- Sample Platform domain configured at `platform:7001`
- AquaLogic Service Bus domain configured at `alsb:7001`
- Sample Portal application consumer
- Sample producer

For an AquaLogic Service Bus configuration that supports the configuration defined in this example, see the AquaLogic Service Bus/WSRP code sample, available from the AquaLogic Service Bus code samples page on BEA dev2dev:

`https://codesamples.projects.dev2dev.bea.com/`

Example Projects and Folders

This example includes separate configurations for two producers. Although the actual producer is the same for both examples, from the consumer's point of view, the producers are different.

The structure of the sample is divided into three projects—one containing common resources, and two containing resources for two example producers.

Table 6-3 Projects in the WSRP Interoperability Examples

Folder	Description
`wsrp`	Contains common resources that are not specific to any producer.
`producerExample`	Basic example that is the easiest to configure. Folder contains producer-specified resources. See "Producer-Level Monitoring Example" on page 6-12.
`operationExample`	Full example supports the most fine-grained producer monitoring. Folder contains producer-specified resources. See "Operation-Level Monitoring Example" on page 6-21.

Producer-Level Monitoring Example

The basic configuration example (in the `producerExample folder`) is the easiest configuration to implement. This configuration supports the monitoring of a producer in the aggregate (see "Producer-Level Monitoring" on page 6-8), but it does not consider the constituent services or operations.

Implementing this producer-level monitoring configuration involves:

- creating one business service and one proxy service to retrieve the WSDL from the producer
- creating one business service and one proxy service to invoke the producer services

The rest of this section describes the tasks required to implement this producer-level monitoring configuration.

Step 1: Retrieve the WSDL from the Producer

To configure producer-level monitoring, the first step is to create the resources needed to retrieve the producer's WSDL and return it to the consumer. Because the WSDL contains the endpoints of the producer's services, it is necessary to transform them to hide the IP address and port of the actual server. Instead, the addresses must refer to the AquaLogic Service Bus server, and the URIs must match the URIs that the proxy service defines for this producer.

Step 1.1: Create a Business Service

Create a business service to obtain the WSDL from the producer. This resource is specific to the producer, so it must be created in the `producerExample` project. The following table describes the properties of the business service.

Table 6-4 Business Service Configuration Properties

Name	Value	Comments
Service Name	`wsdlSvc`	Any name is allowed.
Service Type	Any XML Service	Consumers usually retrieve the WSDL from the producer using an HTTP `GET` request. Only XML services support `GET`.
Protocol	`HTTP`	Or HTTPS
Load Balancing Algorithm	`none`	`none` is preferable.
Endpoint URI	`http://platform:7001/producer/producer?WSDL`	Although multiple endpoints may be specified for retrieving the WSDL, doing so is of limited benefit.
HTTP Request Method	`GET`	

Step 1.2: Create an XQuery Expression to Construct URLs

All endpoint addresses in the producer's WSDL must be transformed to reflect the AquaLogic Service Bus server address and the proxy service URI values. Because each producer WSDL can have four or more ports defined, it is convenient to create an XQuery expression to simplify the construction of the endpoint locations. The XQuery expression accepts the following three string variables as input and concatenates them together to form a SOAP address element:

- **base URL** for the AquaLogic Service Bus server
- **name** to identify the producer
- **extension** used to differentiate ports for a producer

The following table shows the query definition in the `wsrp` project.

Table 6-5 XQuery Definition in the wsrp Project

Name	Value
Resource Name	`wsrp/addr`

XQuery	`declare variable $baseURL external;`
	`declare variable $name external;`
	`declare variable $svc external;`
	`declare namespace soap="http://schemas.xmlsoap.org/wsdl/soap/";`
	`<soap:address location="{concat($baseURL, $name, $svc)}"/>`

Step 1.3: Create a No-Op Proxy Service

A subsequent configuration task (see "Step 1.4: Create a Common Proxy Service" on page 6-14) requires a service that does nothing. To create this service, define a new proxy service in the `wsrp` project folder with the resource name `nullSvc`. Accept all of the defaults for this service. Configuring this proxy service creates a message flow for the service of an echo node only, which is all that is required for this example.

Step 1.4: Create a Common Proxy Service

Create a proxy service used by consumers to get WSDLs from producers. This proxy service is appropriate for any producer configuration modeled on this basic sample. The example described in this section is only a suggestion—a different approach might better suit the specific requirements of a given implementation. Because this proxy service is not specific to a single producer, it should be created in the `wsrp` project folder.

The approach used in this step requires the administrator to assign each producer a name that is included in part of the URL to retrieve the WSDL. The message flow for the proxy service will extract the name from the URL, use it to locate the business service specific to that producer, obtain the WSDL, and then transform the WSDL to rewrite the endpoints to AquaLogic Service Bus. The proxy service endpoint URI is configured as `/producerWSDL`, and the URL that consumers use to obtain a WSDL is:

```
http://alsb:7001/producerWSDL/producerName
```

where *`producerName`* is the name assigned to the producer by the administrator. In this example, the producer name is `producerExample`.

The following table describes how the proxy service is configured:

Table 6-6 Proxy Service Configuration Properties

Property Name	Value	Comments
Service Name	`producerWSDL`	Any name is allowed.
Service Type	Any XML Service	
Protocol	`HTTP`	
Endpoint URI	`/producerWSDL`	

The message flow for this proxy service consists of a pipeline pair and a route node. The request side of the pipeline pair consists of a single stage whose job is to extract the producer name from the URL and assign it to a context variable. The action is:

```
Assign $inbound/ctx:transport/ctx:request/http:relative-URI to
variable producerName
```

The response side of the message flow is a stage where all of the transformations are performed. Before executing the Replace Actions to transform the WSDL, assign the base URL of the AquaLogic Service Bus server to a context variable to avoid specifying it on every transformation:

```
Assign "http://alsb:7001/" to variable nonSecureBaseURL
```

Because a producer can implement four ports, the proxy service must transform each port. If the producer does not implement a particular port, the XQuery transformation simply does nothing. Because a single endpoint will be defined to handle all WSRP traffic for this producer, the Replace Action uses the `addr` XQuery resource created earlier (see "Step 1.2: Create an XQuery Expression to Construct URLs" on page 6-13) to transform the endpoint to the value:

Table 6-7 Variable Mapping (wsrp/addr)

Property	Setting
`name:`	`$producerName`
`svc:`	`""`
`BaseURL:`	`$nonSecureBaseURL`

The four Replace Actions are defined as shown in the following code listing. The value of *`name`* is replaced with the binding names from the table.

```
Replace
./wsdl:definitions/wsdl:service/wsdl:port[@binding="name"]/soap:addr
ess[starts-with(attribute::location,"http:")]in variable body with
xqTransform(…)

Replace entire node
name
urn:WSRP_v1_Markup_Binding_SOAP
urn:WSRP_v1_ServiceDescription_Binding_SOAP
urn:WSRP_v1_PortletManagement_Binding_SOAP
urn:WSRP_v1_Registration_Binding_SOAP
```

For the first Replace Action, the following User Namespace definitions must be added:

Table 6-8 User Namespace Definitions on Replace Action

Prefix	Namespace
wsdl	http://schemas.xmlsoap.org/wsdl/
soap	http://schemas.xmlsoap.org/wsdl/soap/

Note: Producers created by BEA tools implement an extension service (`urn:WLP_WSRP_v1_Markup_Ext_Binding_SOAP`). This port is not used in this example. It is harmless to leave its endpoint unmodified.

The route node of this message flow consists of a routing table that selects the case based on `$producerName`. For each known producer (this example uses only one producer named `producerExample`), add cases so that each case routes to the correct business service to retrieve the WSDL if the name matches. This example uses the following directive:

```
= "producerExample" Route to wsdlSvc
```

To handle cases in which an unknown producer name is given, add a Default Case that routes to the no-op service (defined in "Step 1.3: Create a No-Op Proxy Service" on page 6-14):

```
Default Route to nullSvc
```

In this example, return an HTTP 404 status code by adding these response actions to the default case:

```
Insert <http:http-response-code>404</http:http-response-code> as last
child of ./ctx:transport/ctx:response in variable inbound
Reply With Failure
```

Step 2: Configure WSRP Service Processing

After the resources needed to retrieve the producer's WSDL have been created, create the configuration resources to handle normal WSRP service requests via AquaLogic Service Bus. The easiest configuration involves creating a single proxy service and a single business service, and then linking them via a message flow that propagates the transport headers that WSRP requires.

Step 2.1: Create the Business Service

The minimal business service required for WSRP is not based on a WSDL—instead, it is created to accept any SOAP message. This approach simplifies configuration and allows a single business process to handle all port types used by WSRP. The trade-off with this approach is that it limits monitoring capabilities. Configure the business service with the following settings:

Table 6-9 Business Service Configuration Settings

Property Name	Value	Comments
Service Name	`producerSvc`	Any name is allowed.
Service Type	Any SOAP Service	
Protocol	`HTTP`	Or HTTPS if the producer was created with `secure="true"`.
Load Balancing Algorithm	`none`	Must be `none`, or session information will be lost across requests if multiple endpoints are defined.
Endpoint URI	`http://platform:7001 /producer/producer`	Multiple endpoints may be defined for simple producers only. If multiple endpoints are defined for complex producers, Invalid Registration faults will occur.

Step 2.2: Create the Proxy Service

The most convenient way to define the proxy service is to create it from the existing business service defined in the previous step. This creates a proxy service with the correct type (`"Any SOAP Service"`, the same type configured in the business service) and also constructs the basic message flow that unconditionally routes messages to the proper business service. The message flow must be edited in a subsequent step. Configure the proxy service using the following settings.

Table 6-10 Proxy Service Configuration Settings

Name	Value	Comments
Service Name	`proxySvc`	Any name is allowed.
Service Type	Any SOAP Service	
Protocol	`HTTP`	Or HTTPS, if desired. The value does not need to match the secure mode of the producer, but it *does* need to match what is returned in the endpoints in the WSDL. This example uses HTTP.
Endpoint URI	`/producerExample`	Any value may be used, but it must match the location return in the WSDL.
Operation Selection Algorithm	SOAP Body Type	Can also use the `SOAPAction` from the transport header.

Step 2.3: Edit the Message Flow

WSRP relies on data conveyed in the transport headers to function properly. In particular, producers will return to consumers any session cookies in the response headers that they expect consumers to supply in subsequent requests. Similarly, producers expect consumers to provide the requested operation in the `SOAPAction` request header.

By default, AquaLogic Service Bus does not copy transport headers from the inbound request to the outbound request, or from the outbound response to the inbound response. The message flow must therefore propagate the required headers both in and out of the business service. Because these transformations are required for every WSRP service, it is convenient to define two common XQuery resources—one for request headers and one for response headers—that extract the correct headers.

For request headers, use the following query.

Table 6-11 Request Header Query

Name	Value
Resource Name	`wsrp/rqstHeaders`

XQuery	`declare namespace ctx="http://www.bea.com/wli/sb/context";`
	`declare namespace tp="http://www.bea.com/wli/sb/transports";`
	`declare variable $in external;`
	`$in/ctx:transport/ctx:request/tp:headers/child::*[local-name()!="Content-Length"]`

The `rqstHeaders` query extracts all transport headers (except `Content-Length`) from the `$in` variable. AquaLogic Service Bus can sometimes reformat the message body so that its length no longer exactly matches the request message. Copying the length from the original request can result in transport errors if the body was modified (such as reformatted).

To copy the inbound request headers to the outbound business service, add the following Replace request action to the message flow:

```
Replace ./ctx:transport/ctx:request/tp:headers in variable outbound
with xqTransform(…)
Replace node contents
Variable Mapping (wsrp/rqstHeaders):
in:$inbound
```

Similar to the request side, the response side defines a common XQuery resource to extract all but the `Content-Length` header from the response returned from the producer.

For response headers, use the following query.

Table 6-12 Response Header Query

Name	Value
Resource Name	`wsrp/rspncHeaders`

XQuery	`declare namespace ctx="http://www.bea.com/wli/sb/context";`
	`declare namespace tp="http://www.bea.com/wli/sb/transports";`
	`declare variable $out external;`
	`$out/ctx:transport/ctx:response/tp:headers/child::*[local-name()!="Content-Length"]`

The following Replace response action in the route node propagates the required headers:

```
Replace ./ctx:transport/ctx:response/tp:headers in variable inbound
with xqTransform(…)
Replace node contents
Variable Mapping (wsrp/rspncHeaders):
out:$outbound
```

Step 3: Test the Configuration

After completing the simple configuration for the producer-level example, activate the changes made in the session. To test the configuration:

1. Retrieve the WSDL from a regular browser window by entering the following URL:

   `http://alsb:7001/producerWSDL/producerExample`

   to get the WSDL for the sample producer.

2. Verify that all of the endpoint URLs (except for the BEA extension service) have been modified to the AquaLogic Service Bus IP address, port, and correct proxy service for the sample.

3. Create a remote portlet in a Portal consumer application, specifying this URL as the address of the WSDL for the producer.

   Use either the WebLogic Workshop or Portal Administration Tool to create the remote portlet. Except for entering a different URL to retrieve the WSDL, the steps to create this portlet are no different from those used to create the portlet not proxied by AquaLogic Service Bus.

4. After the consumer portal is complete, run the application.

When configuring AquaLogic Service Bus, monitoring of any of the components has not yet been explicitly enabled. The procedure to enable monitoring is no different for WSRP services than it is for any other Web service in AquaLogic Service Bus. For each component of interest, select (check) the Enable Monitoring box on the Manage Monitoring page and set the aggregation interval. Consider setting up any alert rules, if applicable. After these changes have been activated, the configuration can be monitored from the dashboard facility of the AquaLogic Service Bus console.

Operation-Level Monitoring Example

The full monitoring configuration example (in the `operationExample folder`) involves configuring AquaLogic Service Bus to monitor all services and operations of a producer (see "Operation-Level Monitoring" on page 6-8). All of the concepts described for the producer-level monitoring example (see "Producer-Level Monitoring Example" on page 6-12) still apply to this example; to simplify configuration tasks, certain elements of that configuration will be copied. The operation-level monitoring example uses the same producer application as the producer-level monitoring example.

The fundamental difference between this example and the producer-level monitoring example is that the operation-level monitoring configuration uses both business services and proxy services that are based on the WSDLs defined by the WSRP standard. This example defines the additional resources to describe the WSRP services and extend the message flows to support monitoring at the operation level.

The rest of this section describes the tasks required to implement this operation-level monitoring configuration.

Step 1: Define WSDL Resources

Import all of the WSRP WSDL definition files, along with the XML schema files on which the definitions depend. All of the files are available as part of the sample code associated with this example, but the standard resource locations are described in the following table.

Table 6-13 WSDL Resource Definitions

Resource Name	Location
`wsrp_v1_bindings`	`http://www.oasis-open.org/committees/wsrp/specifications/version1/wsrp_v1_bindings.wsdl`
`wsrp_v1_interfaces`	`http://www.oasis-open.org/committees/wsrp/specifications/version1/wsrp_v1_interfaces.wsdl`
`wsrp_v1_types`	`http://www.oasis-open.org/committees/wsrp/specifications/version1/wsrp_v1_types.xsd`
`wlp_wsrp_v1_bindings`	`$BEA_HOME/weblogic81/portal/lib/wsrp/wsrp-common.jar`
`wlp_wsrp_v1_types`	`$BEA_HOME/weblogic81/portal/lib/wsrp/wsrp-common.jar`
`xml`	`http://www.w3.org/2001/xml.xsd`
`wsrpWSDL`	`http://platform:7001/producer/producer?WSDL`

Producers generated by BEA Portal extend the standard WSDLs by defining an additional port that allows messages to be sent using MIME attachments. Describing this extension is beyond the scope of this example, but it is still necessary to define these extension resources if the producer WSDL references them. In this example, an optional task is to create a resource for the WSDL used by the producer. After creating these WSDL and XML Schema resources, edit the references in each resource to resolve the dependencies on other resources.

Step 2: Create Business Services

In the producer-level monitoring example, a single business service was created to process all messages for the producer, an approach that worked because the business service was not associated with a WSDL.

This operation-level monitoring example uses the WSDL bindings for each port type implemented by the producer. Because a business service can be associated with only one WSDL port or binding, a separate business service resource must be created for each. A simple producer implements only the required Markup and Service Description interfaces, while a complex producer also implements the Management and Registration interfaces. The services are created identically (except for the service name and types), as shown in the following table.

Table 6-14 Business Service Configuration

Service Name	Service Type
`base`	`WSDL port: operationExample/wsrpWSDL, port="WSRPBaseService"`
`desc`	`WSDL port: operationExample/wsrpWSDL, port="WSRPServiceDescriptionService"`
`mgmt`	`WSDL port: operationExample/wsrpWSDL, port="WSRPPortletManagementService"`
`reg`	`WSDL port: operationExample/wsrpWSDL, port="WSRPRegistrationService"`

For each service, minimally set the attributes listed in the following table.

Table 6-15 Service Attributes for Business Services

Name	Value	Comments
Protocol	`HTTP`	Or HTTPS if the producer was created with `secure="true"`.
Load Balancing Algorithm	`none`	Must be `none`, or session data will be lost across requests if multiple endpoints are defined.
Endpoint URI	`http://platform:7001/producer/producer`	Multiple endpoints may be defined for simple producers only. If multiple endpoints are defined for complex producers, Invalid Registration faults will occur.

Step 3: Create the Proxy Services

Proxy services in this operation-level monitoring example are very similar to the proxy services created for the producer-level monitoring example, with some important differences:

- Just as the business services are based on a WSDL, the proxy services must be based on the same WSDL.

- One proxy service is created for each business service, but each proxy service must have a different URI.
- The configuration must specify which operation is being invoked (described later in this section).

To create the proxy services:

1. Creating the proxy service for the base WSRP service.

   As in the earlier example, create the proxy service using the existing `operationExample/base` business service as the model. This will automatically base the proxy service on the same WSDL binding as the business service, and it will create a message flow with an unconditional route action to the business service. For the Endpoint URI, anything may be used, such as the producer name with the port type abbreviation appended to it (for example, `/operationExampleBase`).

2. Edit the message flow to add the same transformations that were added for the producer-level monitoring example (see "Step 2.3: Edit the Message Flow" on page 6-18) to copy the request transport headers and response transport headers between the consumer and producer.

3. Specify which operation to invoke.

   Normally, in a Route Action that routes to a WSDL-based service, an operation to invoke (by selecting the correct operation from the drop-down menu) is specified. However, each WSRP port implements several operations, and so the configuration requires a routing table with a case for each operation. Each case requires the same transformations to propagate the transport headers.

   Creating all of the transformations in this way might prove to be quite tedious. Fortunately, there is a more convenient approach. Instead of using the drop-down menu, use another transformation to copy the operation from the proxy service to the business service. Configure this transformation by adding an Insert Action to the Request Actions of the message flow:

   ```
   Insert $inbound/ctx:service/ctx:operation as last child of ./ctx:service
   in variable outbound
   ```

The proxy services for the other business services can be created by repeating these steps, although a shortcut can be used to avoid recreating all of the transformations manually. For example, to create the proxy service for the Service Description service:

1. Create a new proxy service using the existing `operationExample/base` proxy service just created as the model. Following this example, use `/operationExampleDesc` for the Endpoint URI.

2. On the Summary Page, click the edit link for General Configuration. The WSDL binding was created using the Base port, so correct that here to refer to the `WSRPServiceDescriptionService` port.
3. Edit the message flow. The route action refers to the base business service. Correct this to route to the `desc` service.

Step 4: Retrieve the WSDL from the Producer

Just as in the producer-level monitoring example, create a service that will retrieve the WSDL from the producer and transform it to hide the actual producer endpoints. The resources created are very similar to those created in the producer-level monitoring sample, but in this example the proxies for each producer have a different URI. The rest of this section describes how to create the resources to retrieve the producer WSDL.

Step 4.1: Create the Business Service

The business service used to obtain the producer WSDL for this example is identical to the resource used in the producer-level monitoring example (see "Step 1.1: Create a Business Service" on page 6-13).

Step 4.2: Create the Proxy Service

Creating a proxy service (named `wsrp/getWSDL`) using `wsrp/producerWSDL` (see "Step 1.4: Create a Common Proxy Service" on page 6-14) as the model. Edit the stage of the Response Pipeline to modify each Replace Action to make the transformation match the Endpoint URI given to the proxies created earlier. In this example, the proxies were created using the producer name with an abbreviated service type appended to it. The `addr` XQuery resource created earlier (see "Step 1.2: Create an XQuery Expression to Construct URLs" on page 6-13) accepts an extension argument to construct the URI location. Simply change that argument to the proper value, as shown in the following table.

Table 6-16 Extension Settings to Construct the URI Location

If `@binding` is	svc arg of `addr` is
`urn:WSRP_v1_Markup_Binding_SOAP`	`"Base"`
`urn:WSRP_v1_ServiceDescription_Binding_SOAP`	`"Desc"`
`urn:WSRP_v1_PortletManagement_Binding_SOAP`	`"Mgmt"`
`urn:WSRP_v1_Registration_Binding_SOAP`	`"Reg"`

Finally, edit the Routing Table in the route node to make the cases correspond to the producers known to the system.

Step 5: Test the Configuration

After completing the configuration, test it to verify that it works correctly. The testing steps for testing the configuration are similar to the producer-level monitoring example (see "Step 3: Test the Configuration" on page 6-20)—the only difference is with the URL used to retrieve the WSDL from the producer:

```
http://alsb:7001/getWSDL/operationExample
```

1. Retrieve the WSDL from a regular browser window by entering the following URL:

   ```
   http://alsb:7001/getWSDL/operationExample
   ```

2. Verify that all of the endpoint WSRP endpoint URLs (except for the BEA extension service) have been changed to correctly refer to the proxy service values on the AquaLogic Service Bus server.
3. Create a remote portlet in a Portal consumer application, specifying this URL as the address of the WSDL for the producer.

   Use either the WebLogic Workshop or Portal Administration Tool to create the remote portlet. Except for entering a different URL to retrieve the WSDL, the steps to create this portlet are no different from those used to create the portlet not proxied by AquaLogic Service Bus.
4. After the consumer portal is complete, run the application.
5. Enable monitoring on the AquaLogic Service Bus components of interest.
6. Use the AquaLogic Service Bus Console to drill down to see message counts and performance statistics on all WSRP services and operations handled by the producer.